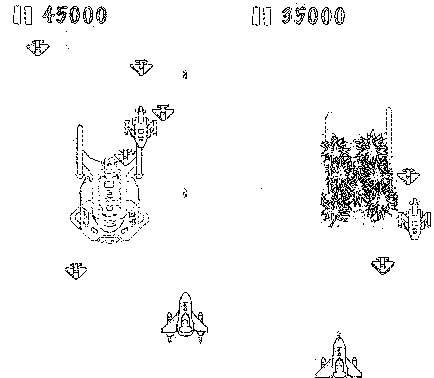
第23章 飞机大战

飞机大战是移动应用兴起之初出现的一款手机射击类小游戏，这款游戏的界面采用手绘 风格，画面简洁有趣，规则简单易懂，操作简便易学，成为风靡一时的亲民游戏。本章使用 Python语言开发一个功能更完善的增强版飞机大战游戏，带领大家一步步体验如何使用前面所 学知识开发一个桌面游戏项目。

濾戏简命 「 .

23JJ fSWte®

经典飞机大战是腾讯交流软件微信5.0版本在2013年8月推出的软件内置小游戏， iPhone. iPad等苹果用户可以前往App Store下载体验，安卓版也在2013年8月推出。下面是 经典飞机大战的游戏界面截图，如图23」所示。



*%.*

经典飞机大战的玩法非常简单，玩家点击并移动自己的英雄飞机，在躲避迎面而来的敌方 飞机时，通过发射炮弹打掉飞机来赢取分数。一旦撞上敌方飞机，游戏就会结束。

本章将实现一个比经典飞机大战功能更丰富的增强版飞机大战游戏，引导读者在实际应用 中正确地使用面向对象思想编程，并通过使用pygame的高级功能，体验使用Pytiioii语言开发 游戏项目的乐趣。

23J.2曜卷願飞飆失戡濺鸚舞慮

增强版飞机大战在经典版飞机大战的基础上增加了若干功能，提高了游戏的趣味性和可玩 性。增强版飞机大战游戏的特点包括：

•英雄飞机自动且连续地发射子弹，不需要玩家手动控制。

•英雄飞机发射子弹的数量，从每次发射1颗增强为连发3颗。

-为英雄飞机引入生命计数（简称“命数”）o英雄飞机在游戏初始时自带3条命；当被 敌机击毁时减少命数；通过击毁敌机得到分数可增加命数。当英雄飞机的命数为0时， 游戏结束。

-整个游戏设置了3个难度。当英雄飞机的得分超过一定范围后，游戏进入下一个关卡， 难度也相应增加。

-为游戏增加了背景音乐和音效。发射子弹、引爆炸弹、英雄飞机升级、飞机爆炸等都有 不同的音效。

增强版飞机大战的精彩游戏界面如图23以所示。

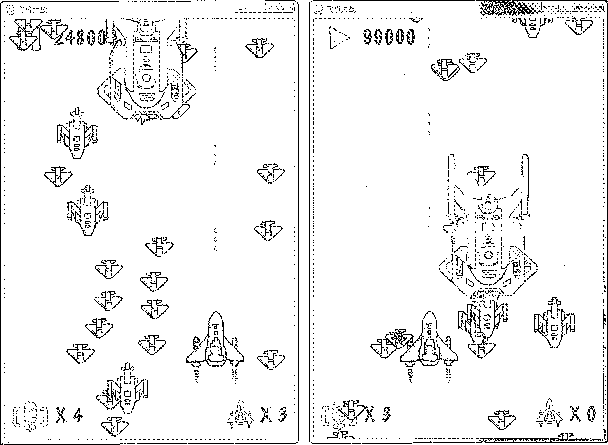
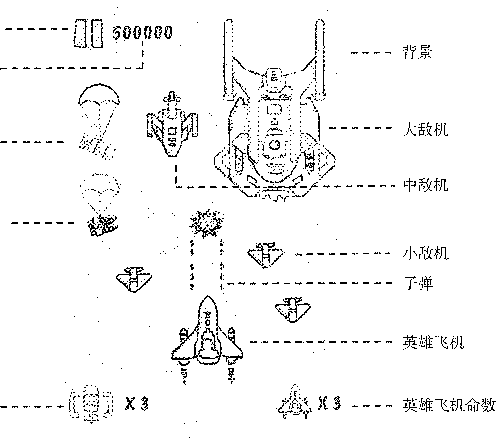


图23-2增强版飞机大战游戏界面

23/L3蹭蠲臓飞縄矣戡避戰谩

增强版飞机大战（以下简称“飞机大战"）的主要元素如图23・3所示



状态图标

分数

炸弹补给道具

子弹增强道具

炸弹数量

图23；飞机大战界面元素

由图23・3可知，飞机大战的游戏元素主要分为几大类，包括背景、英雄飞机、敌机、道具等。 接下来，就为大家一一介绍这些游戏元素以及相关的游戏规则。

1 “篱景

整个游戏窗口的背景是一张星空图像，该背景图像缓缓地向下方持续运动，给玩家产生一 种英雄飞机向上方飞行的感觉。

另外，在游戏过程中会持续循环播放背景音乐，以提升游戏的体验度。

英雄飞机是由玩家控制的飞机，是飞机大战游戏的主角，它的游戏规则主要包括：

(1 )英雄飞机在游戏开始时，出现在屏幕下方的中央。

(2 )英雄岀场后，会有3 s的无敌时间，也就是在这3 s内，任何敌机都无法撞毁英雄飞机。

1. 玩家可以通过方向键([t ]. [ n. [^1. [-^1)控制飞机在屏幕范围内向上方、 下方、左方、和右方移动,移动速度为5。
2. 英雄飞机出场后，每隔0.2 s?会自动连续发射3颗子弹。这些子弹的行为和特征包括： •子弹的速度为12,杀伤力为1。

«子弹从英雄飞机头部的正上方发射，沿垂直方向向游戏窗口的上方飞行。

-发射子弹时，播放发射子弹的音效。

®如果子弹飞行途中，击中了敌机，会对敌机造成伤害。

•如果子弹飞出了游戏窗口，中途没有击中任何一架敌机，那么该子弹会被销毁。

1. 英雄飞机出场后，会默认携带3颗炸弹。炸弹的行为和特征包括：

•玩家按[B]键会引爆炸弹。一旦炸弹被引爆，游戏窗口中所有的敌机都会被炸毁。

•引爆炸弹后，该英雄飞机所携带的炸弹数量减1。

•当炸弹数量为。时，不再允许玩家继续引爆炸弹。

（6） 英雄飞机带有多个动画和音效。当飞机飞行时，显示飞行动画；当飞机被敌机撞毁时， 显示被撞毁动画，并播放撞毁音效；当飞机升级时,播放升级音效。

（7） 英雄飞机具有多条生命。游戏刚开始时，英雄飞机的生命计数（简称命数）为3；当 英雄飞机被击毁时，命数减1；英雄飞机的得分每增加10万分时，命数加1；当英雄飞机的命 数为。时，游戏结束。英雄飞机的命数实时显示在界面的右下方。

3%敌机

飞机大战游戏中一共有3种类型的敌机：小敌机、中敌机和大敌机，这些敌机对应的特征 如表23』所示。

»23-1敌机的类型和特征

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 名称 | 生命值 | 速度 | 分值 | 飞行动画 | 被击中图片 | 被摧毁动画 | 摧毁音效 |
| 小敌机 | 1 | 1 ~7 | 1 000 | 无 | 无 | 有 | 有 |
| 中敌机 | 6 | 1〜3 | 6 000 | 无 | 有 | 有 | 有 |
| 大敌机 | 15 | 1 | 15 000 | 有 | 有 | 有 | 有 |

从表23』可知，敌机具有生命值、速度、分值、图片和音效等多个特征，并且不同类型 的敌机其特征也不尽相同。

除此之外，还为敌机设计了下列行为：

（1） 敌机岀现在游戏窗口上方的随机位置。

（2） 敌机按照各自不同的速度，沿垂直方向向游戏窗口的下方飞行。

（3） 如果敌机与英雄飞机相撞，则英雄飞机被击毁。

（4） 如果敌机被子弹击中°则敌机的生命值要减去子弹的伤害度，此时根据敌机的生命 值分如下两种情况进行处理:

•如果敌机的生命值大于0,则显示被击中图片（如果有），敌机继续向屏幕下方飞行。

•如果敌机的生命值小于等于0,则播放被撞毁动画及被撞毁音效。在被撞毁动画播放过 程中，该敌机在屏幕上不会移动；在被撞毁动画播放完成后，该敌机被设置为初始状态， 跳转到第（1）步继续执行。

（5） 如果敌机飞岀了游戏窗口，中途没有被击毁，则该敌机被设置为初始状态，并跳转 到第（1 ）步继续执行。

要注意的是，正在播放被撞毁动画的敌机，是已经被摧毁的敌机，不能再被子弹击中，也 不能再撞击英雄飞机。

乱道具 .

在游戏过程中，每隔30 s会从游',戏窗口的上方随机位置向下飞出游戏道具。本游戏共设置 两种道具：炸弹补给和子弹增强。这两种道具的描述如表23.2所示。

» 23-2游戏中的道具

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 功能描述 | 速度 | 播放音效 |
| 炸弹补给 | 英雄飞机拾取后，炸弹数量加1 | 5 | 是 |
| 子弹增强 | 英雄飞机拾取后,发射的子弹改为双排,持续时长20秒 | 5 | 是 |

英雄飞机通过与道具碰撞来拾取道具。

5.分数和奖励

当玩家的子弹或炸弹击毁敌机时，会获得与敌机分值相对应的分数。同一局的分数会不断 累加，并再下一局开始时自动清零。玩家获得的分数实时显示在游戏窗口的左上方。

系统会记录玩家历次游戏所得到的最高分，并在游戏暂停和结束时显示在游戏窗口上。

玩家每得到100 000分会奖励1条命，英雄飞机的命数加1,并实时显示在游戏窗口上。

飞机大战游戏一共设立了 3个关卡,依次是关卡1、关卡2、和关卡3,关卡1为起始关卡。 在玩家得到的分数超过当前关卡的最高分之后，游戏会自动进入下一个关卡。关卡越往后难度 越高，出现的敌机数量和种类更多，速度也更快。这3个关卡的具体设定如表23；所示。

表23-3关卡设定

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 分值范围 | 小敌机数量（速度） | 中敌机数量（速度） | 大敌机数量（速度） |
| 关卡1 | < 10 000 | 16 (1〜3) | 0 (1) | 0 (1) |
| 关卡2 | < 50 000 | 24 (1〜5) | 2"—II | 0 (1) |
| 关卡3 | >=50 000 | 32 (1 〜7) | 4 (1 〜3) | 2 (1) |

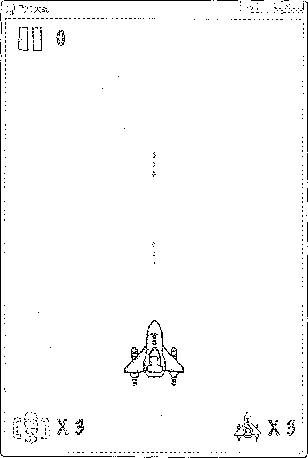
23J.4踞墨牖飞綱我战輿製踢最

飞机大战游戏的典型事件组成了各个典型场景，除了 游戏进行中的场景之外，还包括游戏开始、飞机碰撞、游 戏暂停、游戏结束等多个场景。接下来歹就对这些场景的 设计一一进行介绍。

牝游蔻斑嬌

在游戏开始时，显示的游戏窗口界面如图23-4所示。

在开始界面中，在游戏窗口中间靠下位置显示英雄飞 机，这是游戏的主角，具备移动自身位置、发射子弹、'引 爆炸弹以及拾取道具等功能。游戏窗口右下角显示小飞机 图像x （数字A），其中数字A表示英雄飞机当前的命数； 游戏窗口左下角显示炸弹x （数字B）,其中数字B表示 英雄飞机拥有的炸弹数量；游戏窗口左上角显示1个游戏 状态图像（暂停）和英雄飞机的当前得分。

图23-4所示界面中的元素，除了飞机和背景元素之外， 其他的元素位置都是固定的，但是显示的图像或者数字在游戏过程中可能会发生改变。

么飞机碰瞳 .

如果英雄飞机与敌机发生碰撞，那么对英雄飞机和敌机都要做相应的处理，分别包括下列 内容：

（1） 英雄飞机

发生碰撞时,英雄飞机会被撞毁，此时应播放被撞毁动画及被撞毁音效。在动画播放过程中9 不允许操作英雄飞机；动画播放完成后，英雄的命数减1。

英雄飞机命数减1以后，如果还有剩余命数，那么在英雄飞机牺牲的位置出现新的英雄飞 机继续战斗（新岀场的英雄飞机有3 $的无敌时间，在无敌时间内，英雄飞机不会被撞毁，也 不会撞毁其他敌机）。

如果英雄飞机没有剩余命数，则游戏结束。

（2） 敌机

撞毁英雄飞机的敌机，同样要播放被撞毁动画及被撞毁音效。在动画播放过程中，敌机在 游戏窗口上的位置不会移动。动画播放完成后，敌机会被设置为初始状态，从屏幕的上方沿垂 直方向向下飞行。

图23峪是英雄飞机被撞毁时播放的动画效果中的一个瞬间画面。

游戏暂停

玩家按下空格键可以暂停游戏，再次按下空格键恢复游戏。暂停状态下的游戏窗口 如图23\*6所示。

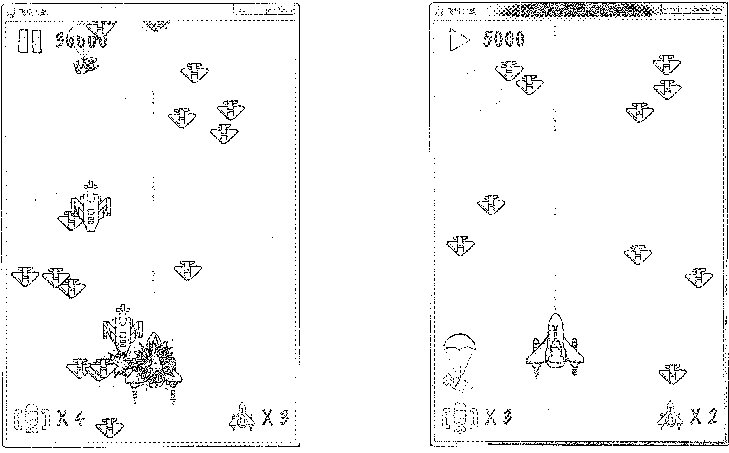


图23-5飞机被撞毁的瞬间 图23-6游戏暂停的画面

在游戏暂停状态下，整个游戏画面静止，同时背景音乐暂停。游戏状态图像更换为运行图 像;界面中央靠上位置显示"Game Paused ! 文字;"'Game Paused!文字下方显示最高得分,玩家可以对比左上角显示的当前得分；最高得分下方显示 "Press spacebar to continue/5 ,提示玩家按下空格键可以继 续游戏。

玩家再次按下空格键恢复游戏后，整个游戏画面恢复罗 同时继续播放背景音乐；游戏状态图像更换为暂停图像； 游戏界面中央位置的提示文字隐藏。

如果英雄飞机没有剩余命数，无法再继续战斗则游戏 结束。在英雄飞机被撞毁的动画播放完成之后，整个游戏 画面静止，并显示如图23・7所示的画面。

其中，游戏窗口中央靠上位置显示“Game Over!”文字。 "Game Over!"下方显示最高得分,玩家可以对比左上角 显示的当前得分。最高得分下方显示uPress spacebar to play again? ?提示玩家按下空格键可以再来一局。

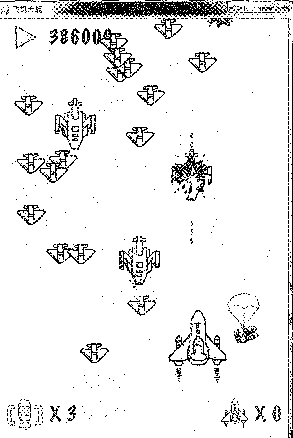
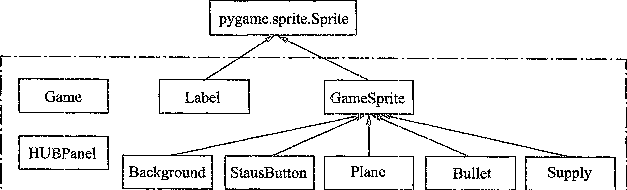
在明确了游戏规则之后，并不能马上进行开发工作，还需要先进行分析与设计等准备工作, 以明确项目的实现方式和内部结构等。接下来就为大家介绍飞机大战游戏项目的准备工作，包 括类设计、模块设计和创建项目结构等。

图23刀游戏结束画面

23.2J美谩諦

由于面向对象开发思想具有结构清晰，易维护等特点，是现如今最常用的开发思想，所以 本项目将使用完全面向对象的设计思想和实现方式。

面向对象设计中最重要的是类的分析与设计，方法是分析程序中出现的各种事物，根据它 们的特征提炼和总结岀程序中需要用的类型。在飞机大战游戏中，我们提炼出的类型及其继承 关系如图23・8所示。

MusicPlayer

Hero

L

Enemy

ij

这些类型中的每一个都有其相对独立的、明确和具体的责任，例如G迎k类负责游戏流程， MusicPlayer类负责音乐播放等。而需要显示在游戏画面上的元素众多，使用了继承自pygame. sprite模块的Sprite类的一系列类型来表示,这一系列类型之间还有着继承关系。

项目中所有类型的简介如表23V所示。

表23-4项目中的类型

|  |  |
| --- | --- |
| 类名称 | 含 义 |
| Game | 游戏类，负责整个游戏的流程 |
| HUBPanel | 指示器面板类，负责统一管理游戏状态、游戏分数、炸弹提示、生命条数以及文本提示等与游戏数据 或状态相关的内容 |
| MusicPlayer | 音乐播放类，专门负责音乐和音效的播放 |
| GameSprite | 游戏的精灵基类，继承自pygame.sprite模块的Sprite类 |
| Background | 游戏背景类，负责显示游戏的背景图像。继承自GameSprite |
| StatusButton | 状态按钮类，负责显示游戏的状态按钮。继承自GameSprite |
| Plane | 飞机类，表示游戏中的飞机。继承自GameSprite |
| Enemy | 敌机类，表示游戏中的敌机。继承自Plane |
| Hero | 英雄飞机类，表示游戏中的英雄飞机。继承自Plane |
| Bullet | 子弹类，表示英雄飞机发射的子弹。继承自GameSprite |
| Supply | 道具类，负责管理炸弹补给和子弹增强道具。继承自GameSprite |
| Label | 文本标签类，负责显示游戏窗口上的文本。继承自pygame.sprite模块的Sprite类 |

由于项目中使用的类型众多，大家在这里对这些类型有一个整体了解即可。每个类型的详 细设计将在后续章节中陆续介绍。

23n2n2滇爍谶谤

在设计程序时，要考虑到程序的设计理念和设计思想，除此之外，还要考虑项目的结构。 结构设计的原则是代码模块化，容易扩展和维护。为此，我们将项目分为如表23〈所示的4个 模块。

表23-5项目模块设计

|  |  |  |
| --- | --- | --- |
| 模块编号 | 模块文件名称 | 功能或含义 |
| 1 | game.py | 游戏主模块，封装Game类并负责启动游戏 |
| 2 | game\_items.py | 游戏元素模块，封装英雄飞机、子弹、敌机、道具等游戏元素类，并定义全局变量 |
| 3 | game\_hud.py | 游戏面板模块，封装指示器面板类 |
| 4 | game\_\_music.py | 游戏音乐模块，封装音乐播放器类 |

从表23-5可以看出，整个项目分为4个模块，每个模块都有自己明确的相对独立的职责, 这样的划分，比使用单一文件在结构方面更清楚。

23.2.3 'ailSBfflKBWiSB

项目中的模块明确之后，接下来就可以开始创建项目，建立模块以及准备资源文件了，步 骤如下？

| 官 大厲，，— ..  res  > font  images  *w* sound  game\_hud.py gansejtsrfis.py 壹，game music.py |
| --- |

(1 )使用PyCharm新建一个项目，取名为“飞机大战"，使用的解释器为Python 3.6O (2 )在项目中依次建立4个模块:game.py,

图23-9项目文件结构图

game\_items.py^ game\_hud.py 和 game\_music.pyo

1. 将项目随附的资源文件夹res复制到飞机 大战目录下。res目录中包含有三个子目录images. sound和font,分别保存着游戏中需要使用的图片、 声音、和字体素材等资源。

创建成功的项目文件结构如图23-9所示。

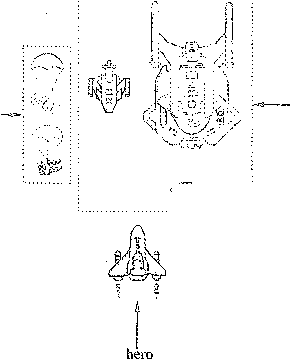
23o3灘戏框禦的険她与簷建

对项目的设计和结构有了整体规划之后，接下来就可以进入项目实现阶段。实现项目的第 一步一般是搭建游戏框架，先将框架搭建起来之后，再向框架内填充具体内容。在本项目中搭 建游戏框架的主要内容包括：

1. 设计游戏(G/me)类，实现游戏类的基础代码，创建并启动游戏主窗口。
2. 利用游戏循环监听退出事件。

完成这些内容，就能为后续的代码演练及开发做好准备和铺垫。

23.3J

游戏类(Game)的功能是负责整个游戏的流程，同时也要包含游戏中的主要元素，基于 这个思想设计之后的游戏类，其类图如图23』0所示。

Game"

英雄精灵

suppliesgroup

道具精员组

enemies\_group

敌机精员组

mam\_window 游戏主窗口 is\_game\_over缉東标记 is\_pause暂停标记 here英雄精冕 hud\_panel指示器面板 典炉音乐播放器

all-group所有魅嚮m屈 enemies group敌机精灵组 supplies\_group道具精灵组

reset\_game重置游戏

create\_enemies 创建敌机 create二supplies 创建道具

start开始游戏 event handler事件监听 checKZcollide 碰撞癥測

图2340游戏类(Game，)的设计图

的属性 . '「

根据属性的作用,我们把游戏类的属性分为2类：游戏属性和精灵组属性。

(1 )游戏属性

在游戏对象中，除了常规的主窗口属性和游戏状态属性之外，还定义了 2个与精灵相关的 属性以及1个音乐播放器属性。在游戏开发中，通常把显示图像的对象称为精灵(Sprite)。在 pygame中精灵有2个重要的属性：

1. image属性:要显示的图像(Suifg对象)。
2. rect属性：图像要显示在游戏窗口的矩形区域。

我们在游戏窗口中看到的每一个独立的图像或者一行文本，都可以看作是一个精灵Sprite 对象，例如：英雄飞机、一颗子弹、一架敌机、分数标签等。

游戏类中定义的游戏属性列表如表23-6所示。

|  |  |  |
| --- | --- | --- |
| 贷23书游戏属性列表 | | |
| 序号 | 名 称 | 说 明 |
| 1 | main\_window | 游戏主窗口，初始大小为(480, 700) |
| 2 | is\_game\_over | 游戏结束标记，初始为False |
| 3 | is jpause | 游戏暂停标记，初始为False |
| 4 | hero | 英雄精灵，初始显示在游戏窗口中间靠下位置 |
| 5 | hudjianel | 指示器面板，负责显示和游戏状态以及数据相关的内容,包括：状态图像、游戏得分、炸 弹数量、英雄命数，以及在游戏暂停或结束时，显示在游戏窗口中央位置的提示信息等 |
| 6 | player | 音乐播放器，负责播放背景音乐和游戏音效 |

(2)精灵组属性

顾名思义，精灵组就是保存了多个精灵对象的组。在pygame中，精灵组有2个非常重要 的应用场景：

1. 一次性绘制或者更新多个精灵。
2. 碰撞检测。所谓碰撞检测就是检测多个精灵之间是否发生碰撞，例如：子弹是否击中敌 机、敌机是否撞到英雄精灵等。

游戏对象中定义的精灵组属性列表如表23・7所示。

|  |  |  |
| --- | --- | --- |
| > 23r7精灵组属性 | | |
| 序号 | 名称 | 说 明 |
| 1 | all\_group | 所有精灵组，存放所有要显示的精灵，用于屏幕绘制和更新位置 |
| 2 | enemies\_group | 敌机精灵组，存放所有敌机精灵对象，用于检测子弹击中敌机及敌机撞击英雄 |
| 3 | supplies\_group | 道具精灵组，存放所有道具精灵对象，用于检测英雄飞机拾取道具 |

2.游戏类的方法

在游戏类中封装了多个方法，用于创建游戏元素，管理游戏的流程，监听系统事件等。游 戏类的方法列表如表23-8所示。

表23书游戏类的方法

|  |  |  |
| --- | --- | --- |
| 序号 | 名 称 | 说 明 |
| 1 | reset\_game | 重置游戏，开启新一轮游戏之前，将游戏属性恢复到初始值 |
| *2* | create\_enemies | 创建敌机，在新游戏开始或者关卡晋级后，根据当前游戏级别创建敌机精灵 |
| 3 | create\_supplies | 创建道具，游戏开始后每隔30 s随机投放炸弹补给或子弹增强道具 |
| 4 | start | 开始游戏，创建时钟对象并且开启游戏循环，在游戏循环中监听事件、更新精灵位置、 绘制精灵、更新显示、设置刷新帧率 |
| 5 . | event\_handler | 事件监听，监听并处理每一次游戏循环执行时发生的事件，避免游戏循环中的代码过长 |
| 6 | check\_colli.de | 碰撞检测，监听并处理每一次游戏循环执行时是否发生精灵与精灵之间的碰撞，其中包 括：子弹击中敌机、英雄拾取道具、敌机撞击英雄等 |

233.2邇赐疆槊躇還

游戏类的设计明确之后，接下来就可以进行游戏框架的搭建工作，包括在gamejtems.py 模块中定义游戏窗口矩形区域的全局常量，在gamely文件中定义Game类，在主程序中创建 游戏对象并且启动游戏等。实现步骤如下：

k窟义塗局变11

在game\_items.py模块中定义全局变量SCREEN\_RECT\ 质是一个矩形区域。代码如下：

用来表示游戏的窗口区域，其本

import pygame

#游戏窗口区域(矩形区域)

SCREEN\_RECT = pygame.Rect(0,0,480,700)

2■实现Game类夔础代码

接下来实现Game类的基础代码，搭建程序的运行流程。

步骤介绍如下:

(1 )定义Game类

在game.py模块中定义Game类,并实现初始化方法( game),代码如下:

.init\_)和重设游戏方法(resRt\_

import pygame

from game\_iterns import \*

from game\_hud import \*

from game\_music import \* class Game(object):

"5游戏类° °

def init (self):

* 1.液主窗口

self.main\_window = pygame.display.set\_mode(SCREEN\_RECT.size) pygame . display. set\_caption ("飞机大战")

* 2 •游戏状态属性

self . is\_game\_over = False # 游戏结束标记

self. is\_pause = False # 游戏暂停标记

* 3.精灵组属性
* 4.创建精灵

|  |  |
| --- | --- |
| def reset\_game(self): |  |
| ° ° °重设游戏"H n |  |
| self o is\_game\_\_over = False | #游戏结束标记 |
| self.is\_pause = False | #游戏暂停标记 |

(2)监听退岀事件

在Game类中定义并实现event\_handler方法,在方法中如果监听到退岀事件则返回True, 否则返回Falseo代码如下：

def event\_handler(self):

° ° "事件监听

:return :如果监听到退出事件,返回True,否则返回False

H ” H

for event in pygame o event.get(): if event.type == pygame.QUIT:

return True

elif event o type == pygame.KEYDOWN and event o key == pygame o K\_ESCAPE: return True

return False

(3)游戏时钟和游戏循环

在Game类中定义start方法，在该方法中定义时钟对象，开启游戏循环，代码如下：

|  |  |  |
| --- | --- | --- |
| 1 | def start(self): |  |
| 2 | ° ° "开始游戏5 “ |  |
| 3 | clock = pygame s time o Clock() | #游戏时钟 |
| 4 | while True: |  |
| 5 | if self o event\_\_handler (): | #事件监听 |
| 6 | return |  |
| 7 | pygame.display.update() | #更新显示 |
| 8 | clock.tick(60) | #设置刷新帧率 |

在上述代码中，第5行调用了 eventjiandler方法监听事件；第7行代码调用了 pygame. display.update方法用于更新显示，只有调用了该方法，才能在游戏窗口中看到最终的绘制结果； 第8行代码设置了刷新帧率，设置之后能够达到流畅的动画效果，并且降低CPU负荷。

1. 荘畫程序审赢动游双 .

在game.py模块的末尾增加以下代码，实现创建游戏对象并且启动游戏的功能，代码如下： if name == ' main 1 :

pygame.init()

Game() .start () pygame.quit()

运行游戏，可以看到游戏主窗口已经创建成功，按【Ese】键或者单击“关闭”按钮都可 以退出程序。

1. 使照窒格键切换游戏状态

调整event\_handler方法，在事件监听的循环体中，增加对空格键按键事件的监听，并在监 听到空格键时切换游戏状态，代码如下：

elif event e type == pygame.KEYDOWN and event.key == pygame B K SPACE:

if self . is\_game\_over: # 游戏已经结束

self.reset\_game()

else : #切换暂停状态

self.is\_pause = not self.is\_pause

接下来修改游戏循环中的代码，使用prim方法输出不同的游戏状态描述文字，如下列代 码中的粗体字所示。

def start(self):

"° °开始游戏"n "

clock = pygame time . Clock () # 游X场时-钟

while True:

if self . event\_handler () : # 事彳牛监听

return

#判断游處状态

if self. is\_game\_\_over: print (n游殲已经结東9按空格健重新开始...°) elif self.is\_pause:

print ("游觀已经暂停歹按空格键继续.O . °) else:

print ("游戏进行中...") pygame <> display. update () # 更糸f显示

clock. tick ( 60) #设置刷新帧率

运行游戏，不断按下空格键，可以看到控制台中有关游戏状态提示文字的显示及变化。比 如某次执行的提示信息如下所示。

游戏进行中。…

游戒已经暂停，按空格键继续。…

游戏进行中…。

如果要测试游戏结束后使用空格键重新开始游戏的功能，可以将初始化方法中的is\_game\_ over属性暂时设置为True,然后再进行测试。

2X4精灵和精灵组

游戏框架搭建好以后，程序已经可以运行，接下来就可以实现显示界面的功能。

从前面的章节我们知道，pygame模块是Python内置的专门用于游戏开发的模块，之前的 游戏项目都是使用pygame实现，我们的飞机大战项目也不例外，也将使用pygame实现。

飞机大战项目需要管理众多游戏对象(例如敌机、英雄飞机、子弹等)，实现众多动画(例 如英雄飞机飞行动画，飞机碰撞动画等)，而使用pygame.sprite模块的精灵类Spr旭和精灵组 类Group来管理这些游戏对象，实现界面元素的显示和动画效果，是再合适不过的。

23n4J Sprite @r©op

为了保证在开发游戏时，能够方便地实现大量图像的绘制工作，pygame专门提供了两个类： 精灵类Sprite和精灵组类Group,它们的类图如图2341所示。

|  |  |  |
| --- | --- | --- |
|  | | Group精灵组 |
| Sprite 精灵 |  | init(\*sprites)初建并添加精灵(多值) sprites()返回组中所有的精灵列表 copy()返回精灵组副本  add(\*sprites)向精灵组添加精灵(多值) remove(\*sprites)从精灵组移除精灵(多值) has(\*sprites)判断是否包含精灵(多值) update(\*args)让组中所有精灵调用update()方法 draw(surface)将组中所以精灵的image绘商在 surface 的 rect 位置  clear(surface,background)将组中所有精灵在 surface的rect矩形区域填充background颜色 empty()清空精灵组所有的精灵 |
| image子类定义  rect子类定义 |  |
| \_init\_(\*groups)创建并添加到指定组 update(^rgs)更新精灵，子类可以根据需求重写 add(\*groups)添加到精灵组(多值) remove(\*groups)从精灵组(多值)中移除 kill()从所有精灵组移除 alive()返回是否包含在精灵组 groups()返回包含精灵的精灵组列表 |  |
|  | |

图23-11精灵和精灵组的类图

精灵类Sprite提供的常用属性和方法如表23-9所示。

表23-9 Sprite类的常用属性和方法

|  |  |  |
| --- | --- | --- |
| 类型 | 名 称 | 说 明 |
| 属性 | image | 记录从磁盘加载的图片内容或者字体渲染的文本内容，注意：子类中必须指定 |
| 属性 | rect | 记录血age要显示在游戏窗口的矩形区域，注意:子类中必须指定 |
| 方法 | update(\*args) | 默认什么都不做，子类可以根据业务需求重写此方法，改变精灵的位置rect或者 显示内容image |
| 方法 | add(5：: groups) | 将精灵添加到指定的精灵组(多值) |
| 方法 | remove(\* groups) | 将精灵从指定的精灵组(多值)移除 |
| 方法 | kill() -> None | 将精灵从所有精灵组移除 |
| 要注意的是，按照pygame官方文档的建议多Sprite类是一个基类，在实际开发中建议先派 生子类再使用。  精灵组类Group提供的常用方法如表2340所示。 | | |

表235。Group类的常用方法

|  |  |  |
| --- | --- | --- |
| 类型 | 名 称 | 说 明 |
| 方法 | update。args) | 让组内所有精灵调用update(\*args)方法，从而实现一次性改变多个精灵的位置 rect或者显示内容image的功能 |
| 方法 | draw(surface) | 将组内所有精灵的image绘在surface的rect矩形区域，从而实现在游戏窗口中一  次性绘制多个精灵的功能 |
| 方法 | sprites() | 返回精灵组中所有精灵的列表 |
| 方法 | add(\* sprites) | 向精灵组中添加指定的精灵(多值) |
| 方法 | remove(\*sprites) | 从精灵组中移除指定的精灵(多值) |
| 方法 | emptyQ | 清空精灵组中所有的精灵 |

简单介绍了精灵和精灵组的概念之后，接下来使用精灵和精灵组实现在游戏主窗口中绘制

游戏背景和英雄飞机的功能。

23.4.2獗箜濾赐精灵琴襲

按照pygame官方文档的建议，我们在实际开发中不直接使用Sprite类，而是先派生Sprite 类的子类再使用。从pygame官方文档可知,在派生Sprite类的子类时有几个注意事项:

•子类可以重写update方法。

®子类必须给image和rect属性赋值。

•子类的初始化方法能够接收任意数量的精灵组对象作为参数，以将创建完的精灵对象添 加到这些指定的精灵组中。

-子类的初始化方法中，必须调用父类的初始化方法，才能向精灵组中添加精灵。

接下来,我们就在game\_items.py模块中派生一个pygame.sprite.Sprite的子类GameSprite, 其类图如图23J2所示。

GameSprite 孔

image 图像 "~

rect矩形区域

speed移动速度

\_init\_(self, image name, speed, \* groups) update(self, \*argsj默认以speed为速度在直方向移动 |

图 23-12 GameSprite 的类图

GameSprite类的实现代码如下所示。

class GameSprite (pygame o sprite <> Sprite):

"° °游戏精灵类° ° n

res\_path = " . /res/images/° # 图片资源路径

def init (selff image\_namef speed, ^groups):

""n初始化方法

:param image\_name :要加载的图片 文件名 :param speed:移动速度，0表示静止

:param groups :要添加到的精灵组r不传则不添加 super ()。 init (^groups)

self <, image = pygame image «load (self o res\_path + image\_name) self.rect = self.image.get\_rect() seslf » speed = speed

#图像

#矩形区域默认在左上角

#移动速度

def update(selff \*args):

"n °更新精灵位置s默认在垂直方向移动

:param args: self « rect.y += self s speed

GameSprite类定义了项目中大部分精灵类的通用特征和功能*，*在项目中作为其他精灵类的 基类来使用。

23.4.3绘制濺邈蘭巖浏爨饋飞飙

接下来按照之前设计的类图，在Gam©类的初始化方法中创建3个精灵组，并且创建背景 精灵和英雄精灵，代码如下：

* 3.精灵组属性

self o all\_group = pygame » sprite e Group()

所有精灵组 敌机精灵组 道具精灵组

self e enemies\_group = pygame.sprite.Group()

self « supplies\_group = pygame.sprite s Group()

* h创良精灵-

#背景精灵’向下方移动

GameSprite("background.png", 1r self.all\_group)

#英雄精灵，静止不动 -

hero = GameSprite("melo png"z 0 7 self » all\_group)

hero.rect.center = SCREEN RECT.center

显示在屏幕中央

然后，在start方法中，修改游戏循环中的代码，在判断游戏状态之后，更新显示之前，加 入负责绘制和更新所有精灵的代码。如下列粗体字代码所示。

#判断游戏状态 if self.is\_game\_over:

print (°游戏已经结束《按空格键重新开始。…“) elif self« is\_pause:

print ("游戏已经暂停,按空格键继续.…”) else :

#更新all\_group中所有精灵内礬 self。all\_\_group。update ()

#绘制all\_group中的所有精灵 self。all\_group. draw (self oinain\_\_window) pygame.display。update()

#更新显示

#设置刷新帧率

clock.tick(60)

运行游戏，可以在游戏窗口中看到星空背景和英雄飞机。并且，星空背景正在缓缓地向下 运动，如图2343所示。

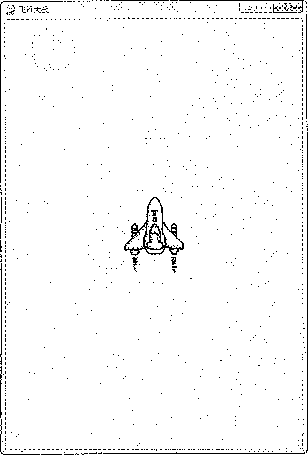


图2343显示背景和英雄飞机

f分多学一招：使用图像与使用精灵和精灵组绘制游戏元素之比较

其实pygame提供了一个专门的图像类Surface,使用这个类也可以实现图像的加载和绘制， 那么为什么在游戏开发中不直接使用图像类而是使用精灵和精灵组呢？这里我们就给大家介绍 一下图像的加载和绘制，大家将使用图像绘制和使用精灵与精灵组开发进行对比，就会理解其 中的原因了。

在游戏中，能够看到的游戏元素可以分为以下3种类型：

1. 图形：是指使用draw模块提供的方法绘制出的形状，例如：圆形、矩形等。
2. 文本：用字体对象渲染指定的文本内容，并生成一张图像(Surface对象)。
3. 图像：将保存在磁盘上的图片文件加载到内存，并生成一张图像(Surface对象)。 要在游戏窗口中看到某一个图片文件的内容，需要以下2个步骤：

(1 )加载图像：使用pygame.image.load()方法从图片文件加载内容并创建图像(Surface 对象)。

(2)绘制图像：调用游戏窗口对象的blit方法将图像绘制到指定的矩形区域，然后调用 pygame.display.update()方法更新显示。

接下来，我们就按照以上2个步骤，分别在游戏主窗口中绘制一下游戏背景和英雄飞机的 图像。

首先,在Game类中定义create\_images方法并实现以下代码。

def create\_images(self):

#加载首景图像

self.background\_image = pygame•image.load(“•/res/images/background.png") self obackground\_rect = self obackground\_image.get\_rect()

#加载英雄图像

self.hero\_image = pygame.image.load("«/res/images/mel.png") self.hero\_rect = self o hero\_image o get\_rect()

#将英雄的总置设置在屏幕中间 ~ -

self o hero\_rect.center = self o background\_rect.center

要注意的是，图片文件的路径务必要指定正确，否则无法正常加载图像数据。

*然后,*在初始化方法末尾调用create images方法,确保在创建游戏对象时能够从磁盘加载 对应的图片文件，代码如下：

#创建图像

self.create\_images()

最后，修改游戏循环的代码，将背景图像和英雄图像绘制在指定矩形区域，代码如下：

#绘制图像

self Omain\_window-blit(self »background\_image, self„background\_rect)

self.main\_window.blit(self o hero\_imagez self o hero\_rect)

运行游戏，可以看到游戏窗口的整个区域被一张星空的背景图片填满，同时在游戏窗口的 中间有一架英雄飞机，如图2314所示。

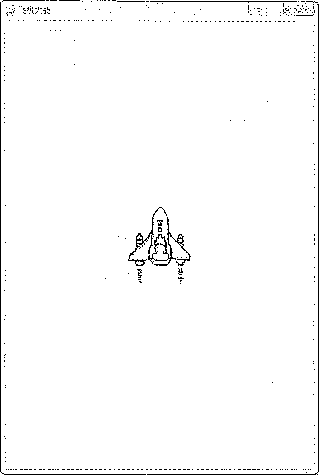
通过上述代码演练不难发现，在pygame中要想显示保存在磁盘上的图片文件还是非常容 易的，只需2个步骤:：在初始化时加载图像，然后在游戏循环中绘制图像。

那么，在实际的游戏开发中，为什么不建议直接使 用这种方式呢？原因在于，为了保证在游戏循环中能够使 用到已经创建的图像对象,我们需要在create\_images方 法中，使用不同的属性记录每一个图像对象及图像对象 的矩形区域。如果游戏中要绘制的图像非常多，那么就 意味着，我们需要在游戏循环中，让游戏窗口对象反复 调用blit方法，依次绘制每一个图像，这是一件非常烦琐 的工作。

而与之相比，使用精灵和精灵组则具有如下所示的 很多优点。

(1)在创建图像对象时，不需要为每一个对象设置 属性。

(2 )在游戏循环中，无论要绘制的图像有多少，只 需要让self.all group调用一下draw方法即可。

(3 )在游戏循环中,如果需要更新游戏元素的内容 图23-14显示背景和英雄飞机

(矩形区域或图像)，只需要让self.all\_group调用一下update方法即可。

显然这些优点对于管理大量的游戏元素是非常方便易用的。

通过前后的对比，相信大家对精灵和精灵组的概念有一个更好的把41,同时也能理解为什 么在游戏开发中应该使用精灵和精灵组而不是直接使用图像了。

在飞机大战游戏运行的时候，整个游戏窗口会显示一张以星空为背景的图像，同时背景图 像会缓缓地不间断地向下方运动，给玩家产生一种英雄飞机向上方飞行的感觉。

但是，我们在运行上一小节实现的程序时发现，游戏刚刚启动时，背景图像是填满整个游 戏窗口的，但是随着背景图像缓缓向下移动，游戏窗口上方会出现一个越来越大的灰色无图片 区域。那么，如何才能实现背景图片连续不断地向下方移动的效果呢？答案是使用2张背景精灵： 背景精灵1和背景精灵2,其实现原理如图23-15所示。

(1)首先将背景精灵1显示在屏幕上，与屏幕重合；背景精灵2放在屏幕的正上方，与 背景精灵1连接起来；

(2 )两个背景精灵同时向下移动;

(3 )当背景精灵1移出屏幕外时，背景精灵2正好全部显示在屏幕上；

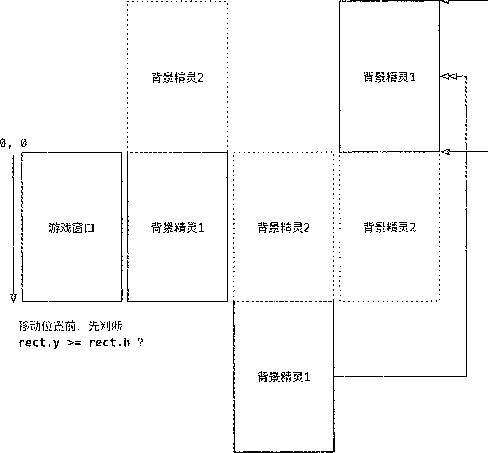
(4 )马上将背景精灵1移动到屏幕的上方，与背景精灵2连接；

(5)两个背景精灵继续一起向下移动，回到步骤(2)。

按照以上步骤，就可以实现背景图像持续向下移动的效果。在判断背景图像是否移出屏幕 下方时，可通过判断图像的y值是否大于等于屏幕的h值来实现；在将背景图像移动到屏幕上 方时，可通过将背景图像的y值设置为图像的高度的负值来实现。

理解了背景图像的实现原理之后，就可以开始设计表示背景图片的背景精灵类

Background,其类图如图 23-16 所ZK。



rect.y = 0

图2345背景图像持续滚动的实现原理

reel.y = ~rect.h

. . 窗am蠱壹t艘. . ..

image图像 rect矩形区域

speed務动速度

updatefself, \*args)钦认以speed再速度在善直方网移动

傳鶴鴛It覆『夺醐聡破

叩date爲\*args)如票移出廢徵 重新设置矩形区域

图2346背景精灵类的-设计

从类图可知，背景精灵类Background继承自GameSprite9并且重写了初始化方法 init 和update方法。在―init 方法中添加了一个参数is alt,用于区别两个背景精灵。is\_alt值为 False第1个背景精灵，其初始矩形区域应该和游戏窗口重叠；为True表示另1个背景精灵， 其初始矩形区域应该在游戏窗口的正上方。

重写update方法,是因为要判断图像是否移出了游戏窗口。如果是，则需要将背景精灵再 次移动到游戏窗口的正上方,从而实现交替滚动。

接下来进行代码实现，首先在代码中创建背景精灵类。打开gamejtems.py文件，添加 BackGround类的定义,代码如下:

class Background(GameSprite):

“5背景精灵类5

def init (self, is\_alt, ^groups)

Fk砺父类方法实碗精灵的创建

super(). init ("background.png", 1r 大groups)

#判断是舌差另精灵,如果是*，*需要设置初始位置

if is\_alt:

self . rect o y = -self rect. h # 设置到游戏窗口正上方

def update(self\*args):

* 1.调用父类的方法实现向下运动

super () .update (\*args) # 向下运动

* 2.判断是否移出屏幕，如果移出屏幕/将图像设置到屏幕的上方 if self o rect. y >= self <» rect h:

self.rect.y = -self.rect.h

然后，修改Game类初始化方法末尾有关创建背景精灵的代码，创建2个背景精灵对象并 添加到精灵组中。修改前和修改后的代码如下:

修改前：

# 4.创建精灵

#背景精灵,向下方移动

GameSprite("background.png", 1, self.all\_group)

修改后:

#背景精灵，交替滚动

self “ all\_group。add(Background. (False)f Background(True))

运行程序，就可以看到背景图片持续往下方移动的效果。如图23』7所示。

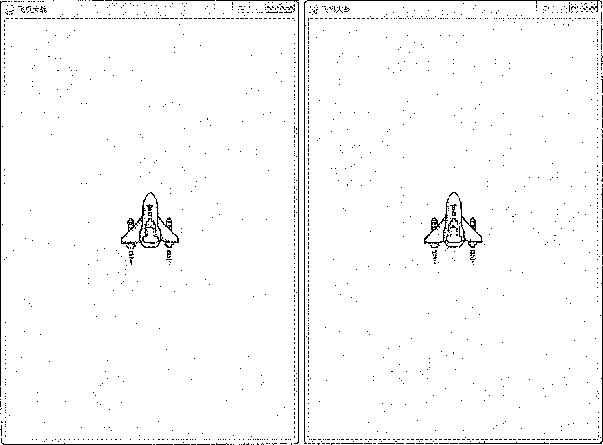


图23J7背景图片持续向下滚动

23O5漏示器菠

指示器面板又称HUD,即Head Up Display (平视显示器)的缩写，它是普遍运用在 航空器上的飞行辅助仪器。在游戏开发中借鉴了这个概念，把游戏相关的信息以类似HUD的方式显示在游戏画面上，让玩家可以随时了解最重要、最直接的游戏信息。

在飞机大战游戏中我们同样设计一个指示器面板，负责显示与游戏状态和数据相关的 内容，包括：状态图像、游戏得分、炸弹图像、炸弹数量、英雄命数，以及在游戏暂停或 结束时，显示在游戏窗口中央位置的提示信息等。

23.5J 赡彖器顾纏鹭齣谶谤

我们给指示器面板类取名为HudPanel,并根据其特征和行为进行类的设计。其类图如 图2348所示。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| score-label得斜标筮 | | | | | | | |
|  | HudPanel | | |  | *癘矇""書* |  |  |
|  | score游戏得分 lives„count,生命计数 level游戏级别 best„score最好成纟费 |  |  | status\_sprite  状每灵 — |  |  |  |
|  | status„sprite状态精灵（暂停/继续） boffib„sprite炸弹龍袈 lives„sprite生命祥数精灵 | |  |  |  |  | status\_label |
|  | score„label緝分标签 bomb„label炸弹计数标签 Uves„Iabel生命计数标蟲 | |  |  |  |  | 状态标蓝  best\_sco relabel |
|  | status„label狀悉标签 best„labet最好成绩标签 tip„label提示标签 |  |  |  |  |  | 最好成绩标签  t,lp„Iabel *提稀签* |
|  | reset\_panel璽暨面板 |  |  |  |  |  |
|  | show^bomb是示炸彌数最 show„lives显示生命计数 increasers co re 增加得分 | |  |  |  |  |  |
|  | load„best„score加载譲好成绩 save„best„score保存最好成绩  panel\_paused面板暫停 paneVresume面板恢复 | |  | bomb sprite  炸霾精灵 — | —« : | *一\*薫"* | li.ves„label  ~生命计数标签 |
|  |  |  |  |  | bomb\_tabel 炸弾\*'數标發 | lives\_sprite 生命wm灵 |  |
|  |  |  |  | 图 2348 | 指示器面板的类图 |  |  |
| 1 J | 層示器面板类的属性 | |  |  |  |  |  |
| 根据属性的作用， | | 我们把指示器面板类的属性分为以下2类：游戏数据属性和精灵属性。 | | | | | |
| （1）游戏数据属性 | | |  |  |  |  |  |
| 指示器面板类中封装了 4个与游戏数据相关的属性，如表2341所示。 | | | | | | |  |
|  |  |  | 表2351指示器面板类的游戏数据属性 | | | |  |
| 名 称 | |  |  |  | 说 | 明 |  |
| score | | 游戏得分，初始为0 | | | | | |
| lives\_count | | 英雄的生命计数，初始为3 | | | | | |
| level | | 游戏级别， | | 初始为1 |  |  |  |
| best\_score | | 最好成绩，保存在record.txt文件中 | | | | | |

（2）精灵属性

指示器面板类中封装了 3个图像精灵和6个标签精灵属性，如表2342所示。

|  |  |  |
| --- | --- | --- |
| 表2X2指示器面板类的精灵属性 | | |
| 序号 | 名 称 | 说 明 |
| 1 | status\_sprite | 状态精灵，显示在游戏窗口左上角 |
| 2 | bomb\_sprite | 炸弹精灵，显示在游戏窗口左下角 |
| 3 | lives\_sprite | 生命计数精灵，显示在游戏窗口右下角 |
| 4 | scorejabel | 得分标签，显示在状态精灵的右侧 |
| 5 | bomb\_label | 炸弹计数标签，显示在炸弹精灵的右侧 |
| 6 | lives\_labei | 生命计数标签，显示在生命计数精灵的右侧 |
| 7 | best\_\_label | 最好成绩标签，显示在游戏窗口中间 |
| 8 | status\_label | 状态标签,显示在最好成绩上方,显示Game Over!或者Game Paused! |
| 9 | tip\_label | 提示标签,显示在最好成绩下方,游戏暂停或者结束时,显示Press spacebar to continue/  play again |

要注意的是，指示器面板类只负责游戏数据维护和面板精灵的创建，而面板精灵的绘制 以及更新是由游戏循环中的代码来调用的。因此我们可以给指示器面板的初始化方法增加一个 display\_group参数，用于传递负责更新或绘制的精灵组对象。这样，在初始化方法执行时，就 可以将创建好的精灵添加到这个精灵组，然后在游戏循环中统一绘制和更新。

2a指示器面板类的有達

指示器面板类中封装了众多负责显示和更新数据和游戏状态的方法，如表23D3所示。

表23-13指示器面板类的方法

|  |  |  |
| --- | --- | --- |
| IlllOsilll | 名 称 | 说 明 |
| 1 | reset\_panel | 重置面板数据，开启新一轮游戏之前，将游戏数据属性恢复到初始值 |
| 2 | show\_bomb | 显示炸弹数量，根据传入的炸弹数量，更改炸弹计数标签显示 |
| 3 | show\_lives | 显示生命计数，使用lives\_count属性值更新生命计数标签显示 |
| 4 | increase^score | 增加分数，根据消灭的敌机分值，增加游戏得分、计算是否奖励生命、调整最好成绩以 及计算游戏级别，并且更改分数标签显示 |
| 5 | load\_best\_score | 加载最好成绩，从record.txt加载最好成绩 |
| 6 | save\_best\_socre | 保存最好成绩，将最好成绩保存到record.txt |
| 7 | panel\_paused | 面板暂停，在游戏暂停/结束时，在游戏窗口中间位置显示游戏暂停/结束的提示信息 |
| 8 | ! panel\_resume | 面板恢复，在游戏执行期间，隐藏游戏窗口中间位置显示的提示信息 |

23.5.2

明确了指示器面板类的设计之后，接下来我们要完成指示器面板类的准备工作，包括创建 指示器面板类，设计状态按钮类，以及完成图像精灵的创建。

们创建指示器面板熒

在game hud.py模块中定义HudPanel类,并实现初始化方法,代码如下:

import pygame

from game\_iterns import \*

class HUDPanel(object):

° ° °指示器面板类心顷

def init (self, display\_group):

5，，初始化方法

:param display\_group:面板中的精灵要被添加到的显示精灵组

#1«游戏属性

|  |  |
| --- | --- |
| self o score = 0 | #游戏得分 |
| self olives\_count = *3* | #生命计数 |
| self »level = 1 | #游戏级别 |
| self.best\_score = 0 | #最好成绩 |

要注意的是，指示器面板类本身并不是一个精灵对象，它只是一个负责管理游戏数据及多 个精灵的面板。所以，HUDPane 1的父类是object,而不是pygame.sprite.Spriteo

1. 设计状态按钮类

按照游戏规则描述，玩家按下空格键暂停或继续游戏时，左上角状态精灵的显示图像会发 生相应的变化。为方便后续的代码编写，我们先在game\_items.py文件中，从GameSprite类派 生一个StatusButton类，专门处理状态图像的切换，其类图如图23J9所示。

&密冊密$快冒土鸞範

image图像

rect矩形区域

$p能d移动溥度—. 」 .. ……

update(5夙们 枱卬摂 默认以 卵鳳關 洲谖度在繇直方向移动

$世翅耄n簽容欺電豎虞註 「「―刀  
扑馅ges图攧列表

3witchf$ta~tus(sel.f, is宀pause)切换状态 |

图2349状态按钮类图

StatusButton类的images属性用于记录状态按钮需要使用的两个图像，switch\_status方法 可以根据is\_pause参数设置按钮应该显示的图像。

接下来进行代码实现。在game\_items.py模块中创建继承自GameSprite类的StatusButton类, 代码如下所示。

class StatusButton(GameSprite):

5"状态按钮类° ° °

def init (self7 image\_names, \*groups):

° n n初始化方法

:param image\_names :要加载的图像名称列表

:param groups :要添加至寸的精灵组

w n n

super(). init (image\_names[0], 0, ^groups)

#加载图像 一 -

self.images = [pygame.image.load(self.res\_path + name)

for name in image\_names]

def switch\_status(selff is\_pause):

° n n切换状态

:param is\_pause : 是否暂停

n n n

self o image = self.images[1 if is\_pause else 0]

1. 创建图像精灵

接下来，我们在初始化方法中创建指示器面板中的图像精灵，包括左上角的状态精灵(暂 停/继续)、左下角的炸弹精灵，以及右下角的生命计数(小飞机)精灵。

首先，在初始化方法的上方，定义几个类属性，以方便后续设置精灵的矩形区域和标签精 灵的字体颜色，代码如下：

|  |  |
| --- | --- |
| class HUDPanel(object): |  |
| "° °指示器面板类"° ° |  |
| margin = 10 | #精灵之间的间距 |
| white = (255, 255, 255) | #白色 |
| gray = (64 r 64’ 64) | #灰色 |

然后，在初始化方法的末尾增加以下代码，创建精灵并设置显示位置。

* 2。创建图像精灵
* 1＞状态精灵

self . status\_sprite = StatusButton ( ("pause .pngn , "resume ,.png°) , display\_ group)

self.status\_sprite rect»topleft = (self »marginf self.margin)

* 2＞炸弹精灵

self »bomb\_sprite = GameSprite("bomb png"f 0 z display\_group)

self.bomb\_sprite.rect.x = self.margin

self.bomb\_sprite.rect.bottom = SCREEN\_RECT.bottom - self.margin

* 3＞生命计数精灵

self olives\_sprite = GameSprite("life »png", 0 z display\_group)

self olives\_sprite « rect.right = SCREEN\_RECT„right - self smargin

self.lives\_sprite» rect.bottom = SCREEN\_RECT.bottom - self.margin

最后，在Game类的初始化方法末尾创建指示器面板，代码如下：

#指示器面板

self.hud\_panel = HUDPanel(self.all\_group)

运行游戏，可以看到游戏主窗口的左上、左下以及右下分别显示着暂停图像、炸弹图像和 小飞机的生命指示图像，如图23-20所示。

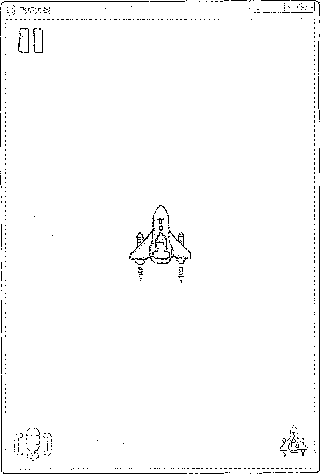


图23-20暂停图像、炸弹图像和小飞机的生命指示图像

23.5.3癒廉精灵靡觐黨搴源篷

接下来我们实现文本标签的相关功能,包括使用自定义字体定义文本标签精灵,以及创建 指示器面板中的文本标签精灵。

1。使照商通义寧係逸义文本标签精灵

之前我们学习过使用pygame的font模块中提供的SysFont类,该类可以创建系统字体对象, 实现在游戏窗口中绘制文字内容。不过，使用系统字体存在以下几点限制：

-不够美观。一般操作系统默认提供的字体大多比较规整，但是用在游戏中往往会感觉有 些呆板。

学不支持跨平台。不同操作系统使用的系统字体的名称不同，这有可能会导致一个应用程 序，被移植到其他类型的操作系统运行时，文字的内容无法被正确显示。

如果在游戏开发时既要做到字体美观，又要支持跨平台，使用自定义字体会是一个不错的 选择。

所谓使用自定义字体就是把字体文件和程序文件保存在同一个目录下，当程序执行时，会 直接加载并使用目录下的字体文件；当程序移植时，会连同字体文件一起被复制。

pygame的font模块还提供了另外一个Font类,可以用来创建自定义字体对象。除了创建 对象的语法略有差异外，其他的用法和SysFont类完全一样。创建Font对象的语法如下:

Font (filenamez size) -> Font

该方法有两个参数，一个是filename,表示要加载的字体文件名或路径；一个是size,表 示字体大小。(在新建项目时9 res/font目录下保存的MarkerFelt.ttc就是我们在飞机大战这个 游戏中需要使用的字体文件。)

接下来,我们就在game\_items.py模块中派生一个pygame.sprite.Sprite的子类Label类,用 于表示文本标签。其类图如图23-21所示。

font字体对象 color文本的颜色 image义本内容渲染生成的图像 rect文本图像矩形区域

5。矽xt (self」： tiexi:)使用；t秘鬟t 靈薪，旦染并重新 『應電t

图23以1 Label类的类图

要注意的是，文本标签的显示内容并不需要在每一次游戏循环执行时都发生变化，因此 不需要重写update。方法。在游戏执行过程中，如果希望更改文本标签的显示内容，可以通过 setjext方法重新渲染文本图像的image属性并且更新矩形区域recto

明确了 Label类的设计之后，就可以进行代码实现了。我们在gamejtems.py模块中定义

Label 类继承自 pygame.sprite.Sprite,代码如下:

class Label(pygame sprite o Sprite):

""n文本标签精灵° " °

f ont\_path = " . / res/font/MarkerFelt. ttc" # 字体文件路径

def init (selfz textz siz㊀/ color, ^groups):

*叫'顷*初始化方法

:param text: 文本内容 :param size : 字体大小 :param color : 字体颜 色 :param groups :要添加到的精灵组

n n n

super() 。 init (^groups)

self . font = pygame fontFont (self . font\_path, size) self o color = color

self ＜, image = self . font. render (textf True ? self . color) self o rect = self.image.get\_rect() def set\_text(self, text):

° ° °设置文本，使用指定的文本重新渲染image,并且更新rect

:param text: 文本内容

H H H

self.image = self.font.render(text, Truef self.color) self o rect = self.image.get\_rect()

z创建指示器面槻宇的文搴标签精灵

接下来，在HUDP観同类的初始化方法末尾增加代码，创建指示器面板中的文本标签精灵, 代码如下。

* 3“创建标签精灵
* 1＞分数标签

self„score\_label = Label("%d" % self.score7 32z self.gray, display\_group) self.score\_label.rect.midleft = (self.status\_sprite.rect.right + *so*丄 self » status\_sprite.rect.centery)

* 2＞炸弹标签 —

32 f self.gray7 display\_group)

self . bomb\_\_label = Label ( °X 3 ° f self ＜.borob\_label B rect midleft =

* 3＞生命计数标签

self »lives\_label = Label ( °X display\_group)

self.lives\_label.rect.midright

#调整生命计数精灵位置

self 8lives\_sprite.rect.right =

* 4＞最好成疏■标签

(self .bom]o\_sprite . rect. right + self .margin^ self.bomb\_sprite.rect.centery)

%d° % self.lives\_countf 32 r self.gray

=(SCREEN\_RECT.right - self.margin,

self.bomb\_label.rect.centery)

self.lives\_label.rect.left - self.margin

self obest\_label = Label("Best: %d" % self.best\_scorez 36z self.white7 display\_group)

self.best\_label.rect.center = SCREEN\_RECT.center

# 5＞状态标签

self.status\_label = Label("Game Over!"f 48, self.white7 display\_group) self. status\_label» rect .midbottom = (self .b€sst\_label. rect. centerx,,

self.best\_label.rect«y 一 2 大

# 6＞提示标签

self.margin)

self.tip\_label = Label("Press spacebar to play again.nf 22, display\_group)

self.tip\_\_label.rect.midtop = (self.best\_label.rect.centerxf self .best label. rect ＜,bottom + 8 \*

在设定标签精灵的位置时有一个小技巧，即参照之前 的游戏屏幕Z5意图9找好每一个标签精灵的参照物，依次 设定即可。也可以设定一个就运行一次，以确保每个标签 精灵的位置都设置正确。

self.white,

运行游戏，可以在游戏主窗口的左上、左下及中间位 置看到对应的文本内容了，如图23・22所示。

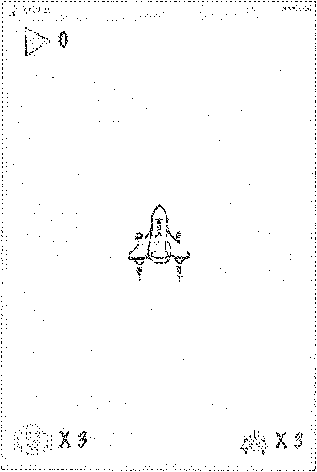
但是，相信大家已经发现了，我们现在完成的指示器 面板还存在以下几个问题没有解决。

•游戏得分、炸弹计数、生命计数显示的数据不会 变化。

°最好成绩始终显示为0o

°提示信息不会随着游戏状态的变化而变化，而是始 终显示在游戏窗口中央位置。

在接下来的小节中，我们就逐一解决这些问题。



self,margin)

图23・22显示文本信息

在这一小节「我们在HUDPanel中封装3个方法，分别实现显示炸弹数量、显示生命计数、 增加游戏得分的功能。每一个方法实现完成后，就在Game的游戏循环或事件监听方法中调用 并测试这些方法。

按照游戏规则描述，英雄出场后，会默认携带3颗炸弹。玩家按下字母b会引爆炸弹。引 爆炸弹后，游戏窗口左下角的炸弹数量相应减少。

要实现实时显示炸弹数量的变化,可以在HUDPanel类中定义show bomb方法,使用传入 的炸弹数量参数更新炸弹计数标签的显示，代码如下：

def show\_bomb(selfz count):

"° n显示炸弹数量

:param count:要显示的炸弹数量

n n n

#设置炸弹标签文字

self\bomb\_\_label. set\_text ("X %dn % count)

#设置炸弹寿签位置

self.bomb\_label.rect omidleft = (self obomb\_sprite»rect 8 right + self.margin^ ' self.bomb\_sprite a rect.centery)

然后,在game.py模块的顶部,使用import random语句导入mmiom模块，以方便使用随机数。 接下来项在Game类的event\_handler方法中添加代码，监听玩家按下字母b事件，并使用 随机数测试炸弹数量的显示。代码如下所示。

#判断是否正在游戏

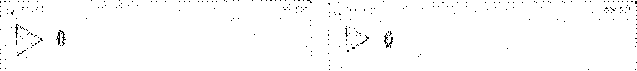
if not self . is\_game\_\_over and not self . is\_pause :

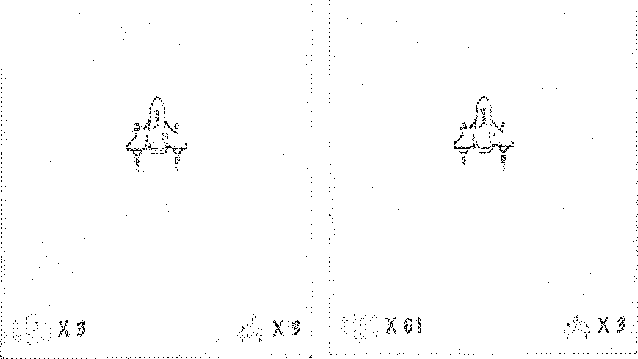
#监听玩家扌安\*字母-b,引爆炸弹 ~

if event.type == pygame.KEYDOWN and event.key == pygame.K\_b:

# TODO测试炸弹数量变化 self.hud\_panel.show\_bomb(random.randint(0f 100))

运行游戏，按下字母b，可以在游戏窗口的左下角看到炸弹数量的变化，如图23-23所示。





2b显示生命计数

按照游戏规则描述，英雄飞机被敌机撞毁后，游戏窗口右下角的生命计数应t亥相应减少； 英雄飞机每得到100 000分会被奖励1条命，游戏窗口右下角的生命计数应该相应增加。

而在指示器面板类中，已经包含有生命计数这个属性。因此，要实现生命计数显示的变化， 可以在HUDPgl类中定义showjives方法，使用生命计数属性值更新生命计数标签的显示即可。 代码如下。

def show\_lives(self):

,,""显示生命计数

n n n

#设置生命计数标签文字

cself . lives\_\_label, set\_text ("X %d" % self . lives\_count)

、设置生命计薮标签位置- —

self.lives\_label.rect.midright = (SCREEN\_RECT.right - self.margin^ self.bomb\_label.rect.centery)

#调整生命计数精灵位置 -

self. lives\_\_sprite . rect right = self«lives\_label o rect.. left - self ^margin 接下来，修改Game类的event\_handler方法,在监听到玩家按下字母b的事件处理中,使 用随机数测试生命计数的显示。如下列代码中的粗体字所示。

#判断是否正在游戏

if not self o is\_\_game\_over and not self , is\_pause :

#监听玩家按孑字母-饥引爆炸弹 -

if event.type == pygame.KEYDOWN and event.key == pygame.K\_b:

* TODO测试炸弹数量变化

self «hud\_panel • show\_\_bomb (random, randint (0 f 100))

* TODO溯I我生命计数数章变化

self o hud\_panel. lives\_\_count = random. randint (0 7 10) self.hud\_panel。show^lives()

运行游戏，按下字母b,可以在游戏窗口的右下角看到生命计数的变化。

亀增加游域得分

按照游戏设计，每当英雄飞机摧毁一架敌机之后，游戏得分就应该相应增加。并且，摧毁 的敌机类型不同要增加的分值也应该不同。

同时需要注意的是，随着游戏得分的增加，还有可能会影响到其他属性的变化，其中包括： (1 )生命计数M英雄飞机每得到100 000分会被奖励1条命。

1. 最好成绩：如果当前游戏得分超过了历史最好成绩，应该以当前的游戏得分作为最 好成绩。
2. 游戏级别：按照游戏规则描述，不同的游戏级别，出场敌机的类型、数量以及速度 都不尽相同，而游戏得分和游戏级别的关系如下：

⑥得分＜ 10 000 (级别=1 )

@得分＜ 50 000 (级别*=2 )*

@得分＞=50 000 (级别=3 )

因此，在这一小节要实现的增加分数方法，除了要完成根据摧毁敌机的分值增加游戏得分属性之外，还需处理相关的生命计数、最好成绩以及游戏级别，并且在最后告诉调用方游戏级 别是否提升，以方便调用方的后续处理。

首先，在HUDPanel类的初始化方法上方，定义几个类属性，以方便后续计算奖励生命和 游戏级别，代码如下：

reward score

100000

level2 score

10000

levs13 score = 50000

#奖励分值

#级别2分值

#级别3分值

然后，在HUDPanel类中定义并实现increase\_score方法，代码如下:

def increase\_score(selff enemy\_score):

w增加游戏得分

:param enemy\_score :摧毁敌机的分值

:return :增加enemy\_scores后,游戏级别是否提升

髯队H

* 1 .游戏得分

score = self«score + enemy\_score

* *2.*判断是否奖励生命 ~

if score // self.reward\_score != self.score // self.reward\_score: self.lives\_count += 1 self.show\_lives()

self.score = score

* 3.最好成绩

self.best\_score = score if score > self.best\_score else self.best\_score

* 4.游戏坂别 — -

if score < self elevel2\_score:

level = 1

elif score < self.Ievel3\_score:

level *= 2*

else :

level = 3

is\_upgrade = level != self.level

self.level = level

* 5.修改得分标签内容和位置

self o score\_labelset\_text("%d" % self.score)

self.score\_labelo rect.midleft = (self status\_sprite.rect 8 right +

self «,margin^ self . status\_sprite . rect. centery)

return is\_\_upgrade

接下来，我们在Game类的游戏循环中增加代码测试increase\_\_score方法，如下列代码中的 粗体字所示。

else :

# TODO测试修改游«>

if selfohud\_panel« increase\_score(100): print (" %dn % scjlf»hud^anel. level)

#更新all\_group中所有精灵内容

self.all\_group.update()

运行程序可以看到，游戏窗口左上角的分数标签会快速地变化，一旦达到奖励分数，游戏

窗口右下角的生命•计数数值也会相应增加。如图23・24所示

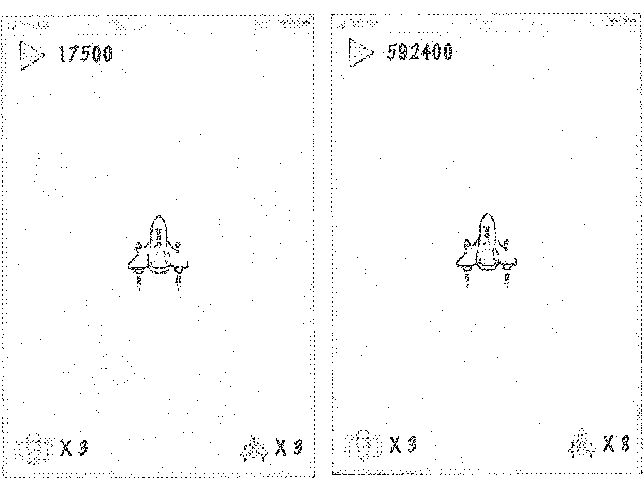


图23・24得分标签和生命计数数值都增加

此外，一旦游戏级别提升，控制面板也会有相应输出，如下所示。 升级到2

升级到3

23n5n5 WWSwWM

最好成绩是一个永久成绩，不随程序退出而消失，所以需要持久化存储。由于存储的数据 量很小，我们使用文件存储的方式就足够了。

在指示器面板类的设计中，设计了 save\_best\_score方法和load best score方法,分别用于 最好成绩的保存和加载，接下来，就实现这两个方法，并在适当的时机调用这两个方法实现最 好成绩的保存和显示。

牝保禧最婦屬绩.

首先，在HUDPanel类的初始化方法上方，定义类属性用于指定保存最好成绩的文件名， 代码如下：

record\_filename = ° record. txt" # 最好成绩文件名

然后,在HUDPanel类中定义save\_best\_score方法,并实现以下代码。

def save\_\_best\_score (self):

”""将最好成绩写入 record. txt"11 °

file = open (self o r㊀cord\_fi1enamep "w°)

file . write ("%d° % self .best\_score) file . close ()

接下来，修改Game类的游戏循环，在游戏退出之前，调用指示器面板的save best score

方法，如下列代码中的粗体字所示。

def start (self):

° ° "开始游戏° ° °

clock = pygame.time.Clock()

while True:

if self.event\_handler():

#游戏时钟

#事件监听

self。hud\_panel。save\_best\_score()

return

运行游戏，待游戏分数增长到一定程度之 后,按［Esc ］退出游戏。就可以在飞机大战 项目目录下，看到一个名为record.txt的文件， 文件内保存着退出之前的游戏分数，如图23・25 所示。

2.旅翱最始斂缱

接下来，我们就在HUDPanel类中定义 load\_best\_score方法,实现加载最好成绩的功

能，代码如下所示。

def load\_best\_score(self):

° ° "从 record.txt 加载最好成绩

性之后,  
字所示。

雾 gee亳 时 O :! S'--察 record.M

*Itt* 潛明 t ［赢《

i res \ i

. ..... game.py

Qame\_hud,py j

gamgjtems^py

*iSji，*game\_mus.k..py \ ;

*聲籌蠶應競诲縱養蕾養鑼蓦靈翊鑼器駿*

External Libraries ::

图23-25保存着最好成绩的record.txt文件

try:

file = open (self . record\_filename) txt = file . readline ()

file . close ()

self.best\_score = int (txt)

except (FileNotFoundErrorf ValueError):

print （"文件不存在或者类型转换错误“）

然后修改HUDPanel的初始化方法，在定义完游戏属 ［；＞ 瀆觥

调用load\_best\_score方法，如下列代码中的粗体| .

def

init (selff display\_group):

EE初始化方法

:param display\_group:

到的显示精灵组-

面板中的精灵要被添加；

#「游戏属性

self o score = 0

self olives\_count = 3

self olevel *= 1*

self obest\_score = 0 self.load\_best\_score()

#

#

#

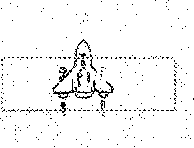
#

游戏得分 生命计数 游戏级别 最好成绩

#加载最好成绩

运行游戏，就可以直接在游戏窗口的中间位置，看到 匸蔑辛

之前保存的最好成绩了，如图23-26所示。 " \*



23.5.6匾源譴残就态

游戏的状态包括进行、暂停和结束这几种，在不同状态下，游戏的指示面板上显示的提示 信息是不同的。这一小节中，就来实现面板暂停和面板恢复的显示功能。其中面板暂停是在游 戏暂停或者结束时，在游戏窗口中央位置显示提示信息，而面板中的其他数据不会再发生变化。

面板恢复是在游戏进行期间，不显示中央位置的提示信息，面板中其他提示标签的数据， 会随着游戏的推进而变化。

精灵组的绘制顺序

指示器面板上的文字应该显示在其他元素之上，也就是说，游戏窗口上的元素显示应该是 有序的，那么应该怎么实现特定的显示效果呢？首先，来观察Game类的初始化方法中创建精 灵的代码，如下所示。

* 4-创建精灵

#背景精灵，交替滚动

self.all\_group。add(Background(False)z Background(True))

#英雄精支，静止不动

hero = GameSprite ("mel. png° f 0 z self . al.l\_group)

hero . rect. center = SCREEN\_RECT . center # 显示在屏幕中央

#指示器面板 —

self. hud\_panel = HUDPanel (self ＜. all\_group)

可以看到9加入精灵组的顺序是背景T英雄精灵一＞ 指示器面板。如果调整代码的顺序，先 将指示器面板加入精灵组，再将英雄精灵加入精灵组，会发现英雄精灵显示在指示文字之上了。

这是因为，在调用精灵组的draw方法时，会按照向精灵组添加的先后顺序来绘制精灵。 也就是，先添加的精灵先被绘制，而后添加的精灵会覆盖在之前绘制的精灵之上。

2实现暂停和恢复面板

无论是暂停还是恢复，面板提示信息必须显示在界面上的其他元素之上，为了保证实现这 种显示效果，我们可以在初始化指示器面板时，只创建3个提示标签精灵的属性，暂时先不将 这3个精灵添加到显示精灵组。

当游戏暂停或结束时，设置3个标签精灵的文字和显示位置，然后将标签精灵添加到显示 精灵组。

当游戏继续时，将3个标签精灵从显示精灵组中移除即可。这样就能保证这3个文本信息 显示在整个界面的最上层。

明确思路之后，我们首先修改HUDPanel类初始化方法中的代码，在创建3个提示标签时, 不传递显示精灵组。修改后的代码如下：

* 4＞最好成绩标签

self.best\_label = Label("Best: %d° % self.best\_score, 36, self»white)

* 5＞状态标签

self.status\_label = Label("Game Over!°, 48’ self.white)

* 6＞提示标銮

self. tip\_label = Label ("Press spacebar to play again . " , 22 7 self white) 此时运行程序,会发现3个提示标签已经看不到了。

接下来,在HUDPanel类中再定义paneljause方法,用于实现面板暂停的操作。该方法 的实现代码如下：

def panel\_pause (self, is\_game\_over, display\_group):..

° "面叛暂停

:param is\_game\_over :是否因为游戏结束需要暂停

:param display\_group : 显示精灵组

#判断是否已经添加了精灵/如果是直接返回

if display\_group.has(self.status\_labelz self.tip\_label, self.best\_label): return

#根据是否结束游戏决定要显示的文字

|  |  |
| --- | --- |
| text tip = | ="Game Over!n if is\_game\_over else "Game Paused!n "Press spacebar to n |

tip += "play again.° if is\_game\_over else "continue. #设置标签文字 — —

selfobest\_label.set\_text("Best: %d" % self.best\_score) self o status\_label.set\_text(text)

self.tip\_label.set\_text(tip)

#设置标礬位置 ~

self »best\_label.rect 8 center = SCREEN\_RECT » center best\_\_rect = self.best\_\_label. rect

self.status\_labelrect.midbottom = (best\_rect.centerx7 best\_rect.y - 2 \* self.margin)

self.tip\_label.rect.midtop = (best\_rect centerxr best\_rect»bottom + 8 \* self.margin)

#添加到精灵组 —

di splay\_group.add(self.best\_labelf self.status\_label^ self s tip\_label) #切换状态精灵状态 — —

self.status\_sprite.switch\_status(True)

然后在HUDPanel类中定义panel\_resume方法，用于实现面板恢复的操作。该方法的实现 代码如下：

def panel\_resume(selfz display\_group): 5“面段恢复

:param display\_\_group :显示精灵组

#从精灵组移除3个标签精灵

display\_group•remove(self。status\_labelz

self.tip\_labelz

self.best\_label)

#切换状态精灵状态 —

se1f « status\_sprite.switch\_status(False)

面板的暂停和恢复方法准备就绪后，接下来修改Game类的游戏循环，在游戏结束和游戏 暂停时，调用指示器面板的panel\_pause方法；在游戏执行时，调用指示器面板的panel\_resume 方法，如下列代码中的粗体字所示。

#判状态

if selfo is\_game\_over:

self. hud\_panel. panel\_pause (True, self <> al Ingroup)

elif self.is\_pause:

self。hud\_panel.panel\_pause(False7 self„alIngroup) else:

self o hud\_panel.panel\_resume(self.alIngroup)

# TODO测试修改游戏得分

if self.hud\_panel.increase\_score(100): print (° 升级到 %d" % self. hud\_panel. level)

#更新all\_group中所有精灵内容

self.all\_group.update()

运行游戏，可以看到游戏窗口中间位置的提示文字，已经可以根据游戏状态的变化而相应 改变了，如图23・27所示。

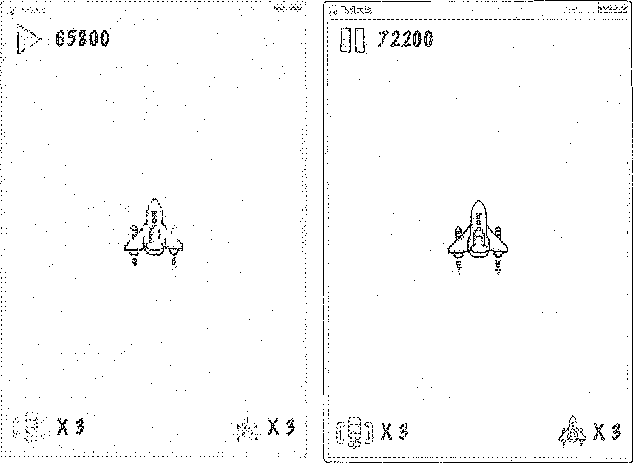


图23-27游戏的提示文字发生变化

23.5.7

当英雄飞机剩余命数为0时，无法再继续进行战斗，游戏结束。此时，如果玩家按下空格 键可以重新开启一轮新的游戏。在新一轮游戏开始之前，我们应该把指示器面板中和游戏数据 相关的属性重新设置，并且更新对应的标签显示，否则会影响到新一轮游戏的数据处理。

n觐斷游戏结案

'在Game类游戏循环的开始位置增加1行代码，判断生命计数是否为0?如果是则认为游 戏结束。调整后的代码如下列代码中粗体字所示。

while True:

#生命计数等于09表示游珑结束

s e 1 f»i s^game\_o ve x = self o hud\_pan@l o lives\_count == 0

if self. event\_handler () : # 事件监听

self.hud\_panel.save\_best\_score() return

2.重置面帼 ’

在新一轮游戏开始之前应该把指示器面板中和游戏数据相关的属性重新设置?并且更新这 些属性对应的标签显示。

为此,在HUDPane 1类中定义resetjane 1方法,并实现以下代码。

|  |  |
| --- | --- |
| def reset\_panel(self): | |
| 心顷重置面板° ° " |  |
| #「游戏属性 |  |
| self o score = 0 | #游戏得分 |
| self.lives\_count = 3 | #生命计数 |
| # 2.标签显示 |  |
| self.increase\_score(0) | #增加0分 |
| self.show\_bomb(3) | #炸弹数量 |
| self.show\_lives() | #生命计数标签 |

然后修改Gam©类的reset\_game方法，在方法末尾添加代码，实现在重置游戏的同时重置 指示器面板，添加的代码如下所示。

self . hud\_panel. reset\_panel () # 重置指示器面板

运行游戏，会发现当3个英雄都牺牲之后，再按空格键可以顺利地开启新一轮的游戏。

邕乱6逐帧动画和飞机类

逐帧动画是一种常见的动画形式(Frame By Frame ),其原理就是在每一次游戏循环执行时, 逐帧绘制不同的内容，从而形成连续播放的动画效果，我们常见的GIF动图就是一种逐帧动画。

逐帧动画在游戏开发中的应用也非常广泛，如图23.28所示。

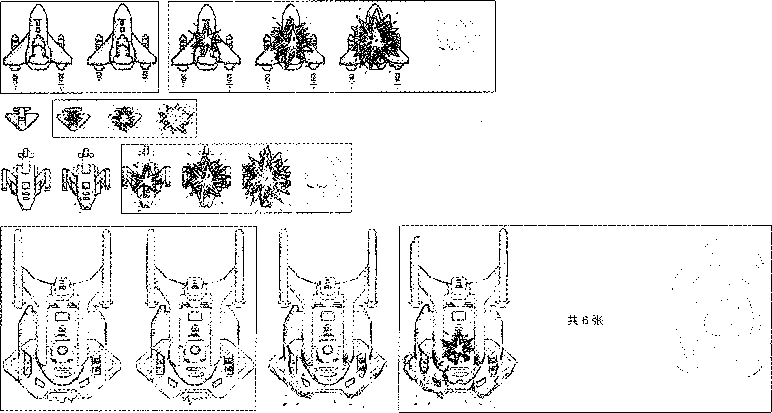


图23・27演示了逐帧动画的实现过程，包括英雄飞机、小敌机、中敌机和大敌机被击毁的 动画实现。通过该图可以看出，在游戏添加这样的动画效果，能大大提高游戏的视觉体验。

在飞机大战游戏中，应根据飞机的不同状态设置对应的图片和动画。飞机状态可以分成3种， 分别是：

(1 )正常飞行状态。

1. 被击中受伤状态。
2. 被摧毁状态。

根据游戏的设定，英雄飞机和小敌机只有1、3两种状态，只有中敌机和大敌机有受伤状态。 这是因为英雄飞机一被撞就会少一条命，小敌机一中枪就会毙命。

不同飞机在各个状态下的显示内容如表2344所示。

表23-14不同飞机的效果设置表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 名称 | 飞行动画 | 被击中图片 | 被摧毁动画 |
| 1 | 英雄飞机 | 有 | *无* | 有 |
| 2 | 小敌机 • | 无 | 无 | 有 |
| 3 | 中敌机 | 无 | 有 | 有 |
| 4 | 大敌机 | 有 | 有 | 有 |

尽管逐帧动画在游戏开发中的应用非常广泛，但是pygame并没有提供直接的实现方式。 在这一小节,先来学习一下如何在pygame中实现逐帧动画。

首先从GameSprite类派生一个简单的飞机类Plane,然后设置逐帧动画频率。

k派生简单的飞机类

根据飞机的特征和行为设计飞机类,取名*为*Plane,并继承自GameSprite类。飞机类的类 图如图23\*29所示。

G讀疆穗$|＞『土土鰹 ... ..「 image图像 rect矩形区域

移动透度 —

jjpdatefself, \*args)默认以speed为速度在垂直方向移动

.一.—...., , …”

norwJindex正常状态图像索引

\_init\_TseIf, norSal^hames"f'''^grouj)7)'

upd霸弋e ( $合If \*)遙薪图像而容

图23-29飞机类的类图

从类图中可知，飞机类中增加了2个属性，这2个属性的介绍如下：

* iionnal\_images属性言记录正常飞行状态的图像列表，而加载每一张图像的图像名称列 表由初始化方法的参数传递。
* normal\_index属性：记录当前要显示的图像索引。

在飞机类中还重写了父类的update方法，在该方法中实现给飞机设置图像的功能。

接下来，就按照Plane类的设计，在game\_items°py模块中定义它的实现代码，如下所示。

class Plane(GameSprite):

”” °飞机类“，顷

def init (selfA normal\_namesf ^groups):

° n °初始化方法

:param normal\_names :正常飞行图像名称列表

:param groups :要添加至”的精灵组

H H H

super() o init (normal\_names[0]z 0胄 ^groups)

#加载图像列表

self . normal\_images = [pygame . image . load (self res\_path + name)

for name in normal\_names]

self.normal\_index = 0

def update(selff \*args):

* 1.设置图像

self o image = self.normal\_images[self.normal\_index]

* 2.更新索引

count = len(self.normal\_images)

self.normal\_index = (self.normal\_index + 1) % count

然后调整Game类的初始化方法，在该方法的末尾添加代码，使用刚刚派生的飞机类来创 建英雄的飞机。添加的代码如下：

#英雄精灵，静止不动

hero = Plane ( [ "me%depng" % i for i in range (1 z 3) ] f self. all\_\_group)

hero . rect. center = SCREEN\_RECT . center # 显示在屏幕中央

运行游戏，仔细观察会看到英雄飞机的尾部会有很细微的火焰喷射效果。之所以要仔细观 *察，*是因为现在完成的代码图像更换的频率太快，导致动画的视觉效果不明显。

2.设置逐帧动画频率

上一小节完成的英雄飞机，飞行动画效果并不是很明显。这是因为我们之前将游戏循环的 刷新帧率设置成了 60帧/s。因为只有将游戏循环的刷新帧率设置成60帧/私玩家在游戏中的 交互才能够得到快速的响应，例如：移动英雄、引爆炸弹等等。如果将刷新帧率设置得太低， 玩家在交互时会感到明显的卡顿。但是，这个刷新帧率的设置对动画显示来说太快。

那怎么样才能做到既保证流畅的用户交互，又降低逐帧动画的动画帧率呢？要解决这个问 题，可以引入一个计数变量。使用它实现运行循环每执行10次才更换一张图像，每秒更换6 次图像，从而达到降低逐帧动画频率的目的。我们先看一下流程图，如图23；0所示。

556 w. Python KS2SSS:映零攣 Python

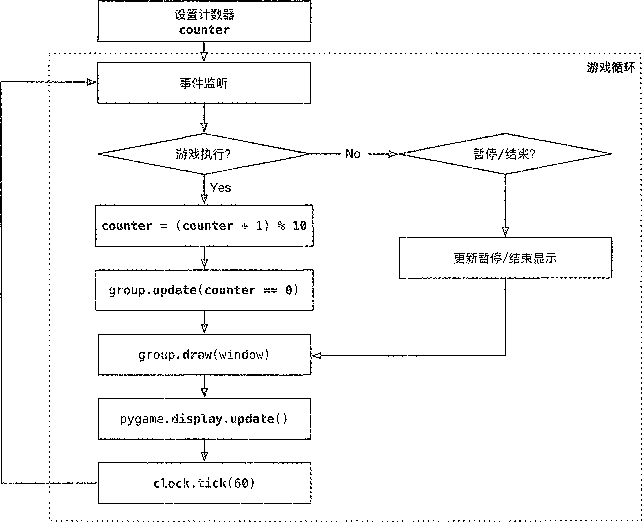


图23-30使用计数变量更新显示的流程图

在该流程图中，使用了算式(counter + 1)%10得到一个余数，只有当余数为0时才会更新 界面显示。

接下来，就按照刚刚明确的思路和流程，在game\_items°py模块的顶部定义一个全局常量， 记录逐帧动画更新的间隔帧数，代码如下：

FRAME\_INTERVAL = 10 # 逐帧动画间 隔帧数

然后修改Game类的stmt方法，在游戏循环开始前定义计数器，代码如下所示。

frame\_counter = 0 # 逐帧动画计数器

在游戏执行期间更新计数器，并将计数器的结果传入update方法。

修改前的代码：

#更新all\_group中所有精灵内容

self„all\_group.update()

修改后的代码：

#修改逐帧动画计数器

frame\_countei2 = (frame\_counter + 1) % FRAME\_INTERVAL

#更新all\_group中所有藉灵内容 一

self.all\_group.update(frame\_counter == 0)

最后，修改Plane的update方法，实现只有在接收到的第1个参数为True时才更换图像， 代码如下：

def update(selff \*args):

#如果第1个参数为False,不需要更新图像，直接返回

if not args[0]:

第23章 飞机大战 *够*557 return

* 1.设置图像

self . image = self . normal\_\_images [self . normal\_\_index]

* 2。更新索引 ― ~

count = len(self.normal\_images)

self.normal\_index = (self.normal\_index + 1) % count

运行游戏，这一次可以非常清楚地看到，英雄飞机的尾部有火焰喷射效果了。

23.6.2飞飜糞韵鰻讲睪蠶IS

学习了 pygame中逐帧动画的实现机制后,在这一小节,就对之前完成的简单飞机类进行

扩展，以便使用Plane类创建出游戏中包含的4种类型的飞机。

1n W飞机类

首先我们将游戏规则描述的4种飞机的全部特征进行总结，以确保在设计时不会有遗漏。

如表23-15所示。

表23-15 4种飞机的全部特征

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 名称 | 生命值 | 速度 | | 分值 | 飞行动画 | 被击申图片 | | 被撞毁动画 | 撞毁音效 |
| 英雄飞机 | 无 | 5 | | 0 | 有 | 无 | | 有 | 有 |
| 小敌机 | 1 | 1〜7 | | 1000 | 有 | 无 | | 有 | 有 |
| 中敌机 | 6 | 1~3 | | 6000 | 有 | 有 | | 有 | 有 |
| 大敌机 | 15 | 1 | | 15000 | 有 | 有 | | 有 | 有 |
| 根据飞机特征的汇总, | | | 进行飞机类(Plane)的设计。其类图如图23・31所示。 | | | | | | |
|  | | | GsdceSprite | | | |  | | |
| image rect範落廛檢  5peed移湖懑鹰 . . | | | |
| ..init(self, image namef speed； ^groups updateTseli / \*args)審秘以speed為懑建在塵観方向整动 | | | |
|  | | |  | |  | |  | | |
|  | | | Pic lie | | | |  | | |
| 郷)蚩前给蠢俺 \*  max^hp初始篷命值  value >« (- vav\_name番效'客  nomal^iniages正常状态囲儼列蔻 normaUindex正常號悉图憾緊引  destroy^, images推獄状忿陸簌列霧 destroy^index繼雾弑态囊弓i | | | |
| hp? speed, value, wav\_namef normal\_naffies.? hurt,„nanie# de§troy„ names f ^groups)  reset„pl^ne(self)愆贸飞瓠 update?sei zgS更新图诲内容 | | | |

图23-31 Plane类的类图

飞机类的属性包括生命值，分值，显示的图像等。它的属性列表如表23』6所示。

« 23-16飞机类的属性列表

|  |  |  |
| --- | --- | --- |
| 序号 | 名 称 | 说 明 |
| 1 | hp | 当前生命值，初始化时指定，被击中时会变化 |
| 2 | max\_hp | 初始生命值,初始化时=当前生命值，用于判断敌机是否受伤0 <： hp < maxjip |
| 3 | value | 分值，敌机被摧毁后的得分 |
| 4 | wav\_\_name | 被摧毁时播放的音效文件名，后续会讲 |
| 5 | normal\_images | 正常状态图像列表 |
| 6 | nomial\_index | 正常状态图像索引 |
| 7 | hurt\_image | 受伤图像(注：所有类型的飞机在受伤时都没有逐帧动画) |
|  | destroy\_images | 摧毁状态图像列表 |
| 9 | destroy\_index | 摧毁状态图像索引 |

飞机类的方法包括重置飞机和更新飞机。它的方法列表如表23-17所示。

表23』7飞机类的方法列表

|  |  |  |
| --- | --- | --- |
| 序号 . | 名 称 | 说 明 |
| 1 | resetjplane | 重置飞机，飞机被摧毁后，重新设置飞机的当前生命值及图像索引 |
| 2 | update | 更新飞机,覆盖父类方法,根据参数设置飞机的显示图像，Plane类中不考虑飞 机位置的变化 |

2.改进飞机类

在这一小节，我们先按照Plane类的设计对飞机类的代码进行改进，然后再使用改进后的 飞机类创建英雄飞机，并测试状态变化时逐帧动画的显示。

(1)改进飞机类

首先对Plane类的初始化方法进行调整，要包含i亥类的所有属性。调整后的代码如下；

def init (self7 hp^ speedy value7 wav\_name7

normal\_namesz hurt\_namef destroy\_names7 ^groups):

n ° n初始化方法

:param hp : 生命值

:param speed: 速度

:param value :得分

:param wav\_name : 音频文件名

:param normal\_names :正常飞行图像名称列表

:param hurt\_name :受伤图像文件名

:param destroy\_names :被摧毁图像文件名

:param groups : 要添加到的精灵组

n n n

super () . init (normal\_names[0]z speed, ^groups)

#飞机属性 — —

self.hp = hp

self.max\_hp = hp

self.value = value

self.wav\_name = wav\_name

#图像属性

* 1＞正常图像列表及索引

self.normal\_images = [pygame.image »load(self.res\_path + name)

for name in normal\_names]

self.normal\_index = 0

* 2＞受伤图處

self o hurt\_image = pygame.image.load(self e res\_path + hurt\_name)

* 3＞被摧成图像列表及索引 ~ *一*

self.destroy\_images = [pygame image oload(self s res\_path + name)

for name in destroy\_names]

self.destroy\_index = 0

然后在初始化方法的下方,定义resetjplane方法,在该方法中重设飞机的生命值以及2个 图像索引，代码如下：

|  |  |  |
| --- | --- | --- |
| def reset\_plane(self): |  |  |
| ° ° "重置飞机""° |  |  |
| se1f » hp = self smax\_hp | # | 生命值 |
| self.normal\_index = 0 | # | 正常状态图像索引 |
| self.destroy\_index = 0 | # | 被摧毁状态图像索引 |
| self.image = self o normal\_images[0] | # | 恢复正常图像 |

然后修改之前完成的Upd砒R方法，实现根据不同的生命值显示不同的图像，代码如下：

def update(selff \*args):

#如果第0个参数为False不需要更新图像，直接返回

if not args[0]:

return

#判断飞机状态

if self . hp == self .max\_hp : # 未受伤

self.image = self.normal\_images[self.normal\_index] count = len(self.normal\_images)

self.normal\_index = (self.normal\_index + 1) % count

elif self .hp > 0: # 受伤

self » image = self . hurt\_\_image

else : #被摧毁

#判断是否显示到最后一张图像，如果是说明飞机完全被摧毁 if self » destroy\_index < len(self.destroy\_images):

self.image = self.destroy\_images[self.destroy\_index]

self.destroy\_index += 1

else :

self o reset\_plane () # 重置飞机

要注意的是，在upda於方法中，当判断到飞机被摧毁后，需要首先判断是否已经显示到最 后一张图像：如果不是，则设置图像并且递增图像索引；如果是，则调用resetjlane方法重置 飞机属性。

(2 )创建并测试飞机对象

接下来在Game类中，使用改进后的飞机类创建英雄飞机，并且测试一下随着生命值的变 化能否切换英雄飞机的逐帧动画。

首先，修改Game类的初始化方法,将创建英雄飞机的代码修改为下列代码。

#英雄精灵*v*静止不动

self . hero = Plane (1000z 5z 07 ??me\_d.own . wav",

["me%d.png" % i for i in range(1r 3)],

"mel.png"f

[°me\_destroy\_%d.. png" % i for i in range (1,5) ] r self"all\_group)

self . hero . rect. center = SCREEN\_RECT . center # 显示在屏幕中央

以上代码在创建英雄飞机时，使用了 selfhero属性来记录飞机对象，这样可以方便在其他 方法中访问到飞机对象。而传入1000作为英雄飞机的初始生命值，这是为了方便稍后的状态 测试。

运行游戏可以看到,运行效果和上一小节完成的代码完全一样,在英雄飞机的尾部有火焰 喷射效果的逐帧动画。

接下来，在游戏循环中增加测试代码，模拟英雄飞机被击毁的场景。通过测试，我们需要 确认，英雄飞机被摧毁后能否正常播放被摧毁的逐帧动画，以及动画播放完成后，英雄飞机能 否被正确地复位。

我们将测试代码添加到修改逐帧动画计算器的正上方，如下列代码中粗体字所示。

# TODO模拟荚雄飞祝受到伤害

self.heroeBp -= 30

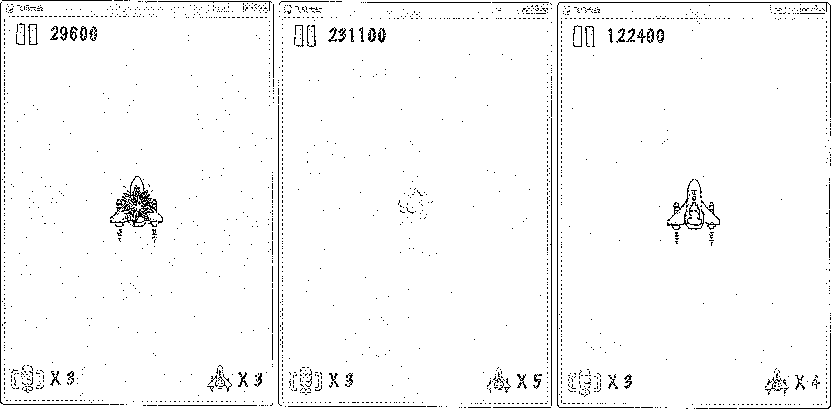
#修改逐帧动画计数器

frame\_counter = (frame\_counter + 1) % FRAME\_INTERVAL

#更新all\_group中所有精灵内容

self o all\_group o update(frame\_counter == 0)

运行游戏，可以看到英雄飞机被摧毁后会播放被摧毁的动画。同时，当动画播放完成后， 英雄飞机能够被正确地复位，如图23・32所示。



23d®d3龌鲤甑郁弄囊 「.

由于敌机和英雄飞机分属于两个不同的阵营，在游戏规则的设定上有不少差异。例如， 敌机初始出现在游戏窗口上方的随机位置，各自以不同的速度飞入游戏窗口，飞出游戏窗口后 会重新设置初始位置及速度等。而英雄飞机在游戏窗口上的位置由玩家来控制。我们有必要以 Plane类为基础，分别为敌机和英雄飞机派生不同的子类。

接下来，首先为敌机派生1个敌机子类，用来表示敌机的特征和行为。

仁谩计敌机曩

在飞机大战游戏中一共有3种类型的敌机，不同关卡出现的敌机数量和速度各不相同。敌 机的初始位置为游戏窗口上方的随机位置，按照各自不同的速度，沿垂直方向向下方飞行进入 游戏窗口。如果敌机从下方飞岀了游戏窗口，中途没有被摧毁，会被设置回初始状态，回到游 戏窗口上方的随机位置。

根据游戏设计，我们对敌机类（Enemy）的特征和行为进行设计，其类图如图23『33所示。

GameSprite

Plane

E踏骚猗y

kind敌机类型而二蔵毎1中敌机、*2*大敌机）

inrt^"（self7n<lj^r"m^"speedr^groupsT

reset\_plane（self）靈置飞机

up四self \*枸rgs）更新图像內容疝頰形区域

图23-33敌机类的类图

|  |  |  |
| --- | --- | --- |
| 敌机类中增加的属性主要包括敌机类型和最大速度，如表2348所示°  表23-18敌机类的属性 | | |
| 序号 | 名 称 | 说 明 |
| 1 | kind | 敌机类型：0—小敌机；1一■中敌机;2———大敌机 |
| 2 | maix\_speed | 最大速度，每次重置飞机时，取1 - max\_speed之间的随机数设置为飞机的速度 |
| 敌机类中需要重写的父类方法包括重置飞机方法和更新飞机方法，如表234 9所示。  » 23-19敌机对象要重新的方法列表 | | |
| 序号 | 名 称 | 说 明 |
| 1 | reset\_plane | 重置飞机，敌机被摧毁或者飞出游戏窗口后，重新设置飞机的位置以及速度 |
| 2 | update | 更新飞机，调用父类方法设置飞机的显示图像，然后根据速度修改飞机的矩形区域， 最后判断是否飞岀游戏窗口，如果是，重置飞机 |

2.实现敌机类的基本功能

接下来实现敌机类的基本功能。首先在gamejtems.py模块的顶部，使用import random语 句导入random模块，以方便使用随机数。

然后按照敌机类的设计思路，在game\_items.py模块中创建继承自Plane类的Enemy类? 实现代码如下：

class Enemy(Plane):

"° n敌机类n ""

def init (self7 kind, max\_speedf ^groups):

n n n初始化方法

:param kind:敌机类型0小敌机1中敌机2大敌机

:param max\_speed: 最大速度

:param groups :要添加到的精灵组

#「记录敌机类型和最大速度

self.kind = kind

self.max\_speed = max\_speed

# 2.根据圭型调用父类方亲传递不同参数

if kind == 0: super() . init (1f 1f 1000 f nenemyl\_down.wav"f

["enemy1.png"]$

"enemyl.png",

["enemyl\_down%dopng" % i for i in range(1f 5)]z ^groups)

elif kind == 1: super() o init (6 z 1z 6000 f °enemy2\_down o wav",

[nenemy2.png"],

°enemy2\_hit.png"z

[nenemy2\_down%d.pngH % i for i in jrange (1 f 5) ] f ^groups)

else :

super() . init (15 z 1r 15 0 0 0 f Henemy3\_down.wav"r

["enemy3\_\_nl spng" r "enemy3\_n2 . png"]*車* "enemy3\_hit.png"f

[nenemy3\_\_down%d«pngn % i for i in i?ange (1 r 7)], \*groups)

# 3.调用重置飞机方法，设置敌机初始位置和速.度

self.reset\_plane()

def re^set\_plane (self):

'顷”重置飞机"5

super().reset\_plane()

#设置初始随机位置和速度

psiss

要注意的是，在resetjlane方法中，暂时没有实现飞机随机位置和速度的设置，后续创建 完敌机精灵后，再来完善此功能。

3話创建敌机精灵

接下来，就在Game类中定义create\_enemies方法，在方法中根据不同游戏级别，创建不

同数量的敌机精灵。关卡和敌机的数量关系如表23^20所示。

S 23-20关卡和敌机的数量关系

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 名称 | 分值范围 | 小敌机数量（速度） | 中敌机数量（速度） | 大敌机数量（速度） |
| 1 | 关卡1 | <10000 | 16 (1~3) | 0 (1) | 0 (1) |
| 2 | 关卡2 | <50000 | 24 (1 〜5) | 2 (1) | 0 (1) |
| 3 | 关卡3 | >=50000 | 32 (1 ~7) | 4 (1 〜3) | 2 (1) |

create enemies方法的代码如下。

def create\_enemies（self）:

“5 根羸游戏级别创建不同数量的敌.机ee

#敌机精灵组中的精灵数量

count = len(self.enemies\_group.sprites()) #要添加到的精灵组

groups = (self.all\_groupf self.enemies\_group) #判断游戏级别及已有时敌机数量 —

if self hud\_panel»level == 1 and count == 0 : # 关卡 1

for i in range(16):

Enemy(0z 3, \*groups)

elif self. hud\_panel. level == 2 and count == 16 : # 关卡 2

* 1>增加敌机的最大速度

for enemy in self.enemies\_group»sprites(): enemy s max\_speed = 5

* 2>创建敌机 for i in range(8):

Enemy(0f 5z ^groups)

for i in range(2):

Enemy*(1r 1f* ^groups)

elif self.hud\_panel. level == 3 and count ==26: # 关卡 3

# 1>增加敌机的最大速度 -

for enemy in self.enemies\_group.sprites():

enemy omax\_\_speed. = 7 if enemy»kind == 0 else 3

# 2＞创建敌机

for i in range(8):

Enemy(0, 7, ^groups)

for i in range(2):

Enemy(lz 3, ^groups)

for i in range(2):

Enemy(2, lz \*groups)

修改Game类的初始化方法,在创建完指器面板之后,调用create\_enemies方法创建敌机, 代码如下:

#创建敌机

self . create\_\_enemies ()

运行游戏，可以在游戏窗口的左上角看到1架敌机，之所以只能看到1架，是因为我们还 没有设置敌机的随机位置。所以第1关的16架小敌机都被绘制在了游戏窗口的左上角的同一 个位置，如图23-34所示。

接下来，我们先修改之前在游戏循环中增加的测试代码，在升级之后调用create\_enemies 方法,以测试游戏级别提升后,能否创建中敌机和大敌机。代码如下:

# TODO测试修改游戏得分

if self.hud\_panel.increase\_score(100): print ("升级至M %d." % self . hud\_panel. level) self.create\_enemies()

运行游戏，当控制台第1次输出升级信息时，游戏窗口的左上角会增加1架中敌机；当控 制台第2次输岀升级信息时，游戏窗口的左上角会增加1架大敌机，如图23；5所示。

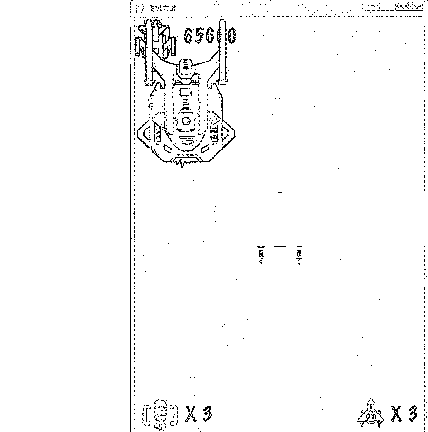
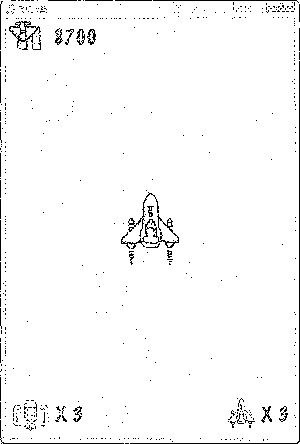


图23-34敌机出现在同一个位置

图23-35升级后创建了 3种飞机

4.谖置敌机精灵的随机位置 .

根据游戏规则，敌机岀现的位置应该是随机的，并且从游戏窗口的上方逐渐进入窗口。接 下来，我们对敌机精灵的初始随机位置进行设置，然后测试当敌机被摧毁后，初始位置是否会 被重新设置。

敌机出现位置的示意图如图23-36所示，根据该示意图可知，设置敌机的初始随机位置的 思路如下：

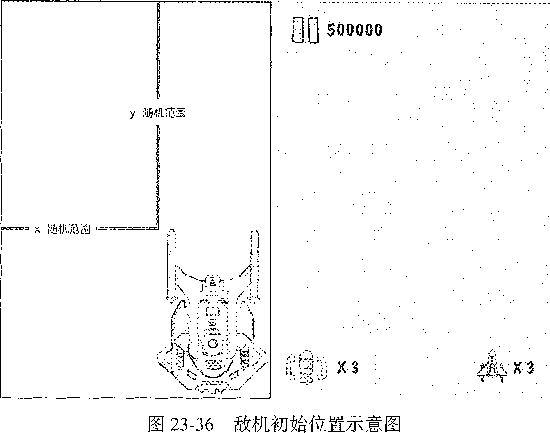
(1 )敌机出现的水平方向坐标取值在0〜(SCREEN\_RECT.w - self.rect.w)之间,可以取 该范围内的一个随机数，作为敌机矩形区域的x值。

(2 )敌机出现的垂直方向坐标取值在0〜(SCREEN\_RECT°h - sRlfecth)之间，可以取该 范围内的一个随机数，再减去游戏窗口高度，作为敌机矩形区域的y值。

这样，就能使敌机初始在游戏窗口上方的随机位置，并且向下飞行进入游戏窗口后，敌机 的边界不会超出游戏窗口的边界。

接下来9我们按照刚刚明确的思路，对Enemy类的reset\_\_plane方法做调整，代码如下：

1 def reset plane(self):



2

3

4

5

6

7

8

° ° °重置飞机叩5

super().reset\_plane() 设置初始随机盅置

= random a randint(0 设置y值

= random o randint(0 7

SCREEN\_RECT.w -self.rect.w)

y

self.rect.topleft = (xz y)

SCREEN\_RECT.h - self.rect.h) - SCREEN RECT.h

为了方便测试敌机的初始位置以及后续的飞机爆炸，我们临时将第7行代码进行更改，改 为只使用随机值，不减去屏幕高度，这样飞机创建出来就都能显示在游戏窗口上了。当本节测 试之后，再把该行代码恢复即可。更改后的代码如下：

y = random. randint (0, SCREEN\_RECT . h - self <. rect. h)

运行游戏，可以看到游戏窗口中首先会出现密密麻麻的小敌机，然后随着游戏级别的提升, 敌机的种类和数量也会相应地增加。

接下来，修改之前在引爆炸弹的事件监听中增加的测试代码，模拟炸毁敌机的场景，测试 当敌机被摧毁后，初始位置是否会被重新设置。修改后的代码如下:

#判断是否正在游戏

if not self . is\_game\_over and not self «. is\_\_pause :

#监听玩家按卡字矽b,引爆炸弹 ~

if event.type == pygame.KEYDOWN and event.key == pygame.K\_b:

# TODO测试炸毁所有敌机

for enemy in self.enemies\_group.sprites():

enemy.hp = 0

运行游戏，按下字母b，可以在游戏窗口看到所有敌机被摧毁的逐帧动画金并且在动画播 放完成后，敌机的位置以及图像都会被重新设置，如图23・37所示。

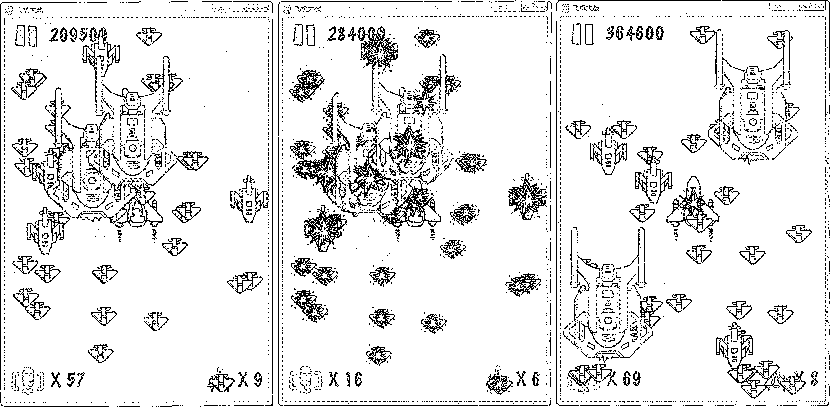


图23-37敌机位置随机生成的效果

亂实觌散机精灵鮑飞行 ，

我们要对敌机的随机速度进行设置，并实现让敌机沿垂直方向向下方飞行，进入游戏窗口； 如果敌机从下方飞出了游戏窗口，会被设置回初始状态。

首先，修改reset\_\_plane方法，根据飞机的最大速度max\_\_speed9设置随机的初始速度，代 码如下：

#设置初始速度

self speed = random„randint(1z self.max\_speed) 然后，重写父类的update方法，实现敌机精灵的飞行功能。代码如下： def update(self7 \*args):

m，更新图像和位置，"5

#调用父类方法更新飞机图像一注意args需要拆包

super () »update(\*args)

#判断敌机是否被摧毁，否则使用速度更新飞机位置

if self ohp > 0:

self,rect.y += self.speed

#判断是否飞出屏幕f如果是r重置飞机

if self.rect.y >= SCREEN\_RECT.h: self « reset\_plane()

在测试代码中，注释掉测试修改游戏得分的代码，如下所示。

* # TODO测试修改游戏得分
* if self.hud\_panel.increase\_score(100):
* print ( n 升级到 %d" % self . hud\_panel. level)
* self o create\_㊀nemies()

运行游戏，由于游戏的初始级别是1,所以会有16架小敌机错落地从游戏窗口的上方徐徐 而来。由于小敌机的速度是随机设置的，而且一旦飞出游戏窗口，会被重新设置到游戏窗口的 上方，所以虽然只创建了 16个小敌机精灵，但从运行效果来看，却会有敌机无穷无尽的感觉，

如图23『38所示。

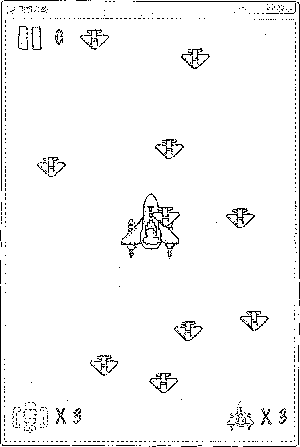


图23郁创建的小飞机

23.6.4 飞飆字榮 「 ..

这一小节中，就以Plane类为基础,派生1个英雄飞机子类。我们只设计英雄飞机子类以 及实现英雄飞机的操控，而有关发射子弹消灭敌机的功能，会在下一小节的碰撞检测中实现。

牝设计襲繼飞机襲’ 、 . \*

游戏规则中关于英雄飞机的描述包括：

（1）游戏开始，英雄出现在游戏窗口中间靠下的位置。

（2 ）英雄出场后，会有3 s的无敌时间?也就是在这3 s内，任何敌机都无法撞毁英雄飞机。

（3） 使用键盘的方向键（gf】、［n . R-】、［->1 ）可以在游戏窗口内移动英 雄飞机，但是需要注意的是：

*會*不能够将英雄飞机移动到游戏窗口之外。

•在播放英雄飞机被撞毁动画过程中，不允许移动英雄飞机或者引爆炸弹。

（4） 英雄出场后，每隔02s，会自动连续发射3颗子弹。

（5 ）英雄出场后，会默认携带3颗炸弹。

-玩家按下字母b会引爆炸弹，一旦引爆，游戏窗口中所有的敌机都会被炸毁。

•引爆炸弹后，游戏窗口左下角的炸弹数量会相应减少，游戏的得分会相应增加。

（6）英雄飞机拾取到子弹增强道具后，发射的子弹改为双排，持续时长20 s。

根据游戏规则，我们设计英雄飞机类（Hero ）得到的类图如图23；9所示°

Plane

「， ，，lte，r\*爲— …

is\_power是否无敌 bomb\_count炸弹数量

buUets„kind子弹类型(0单排1双排｝ bul眞s\_gro叩 子弹精灵缱 \_\_\_\_\_

\_inFE\_(self, ^groups) —

reset„plane(self)竄置飞机 update(selff \*args)翼新图像内容箱組形区域

blowup!self, enemies^group) 爆炸 fire( ? displayfgr-mjp) I寸證i

图23-39英雄飞机类的设计

|  |  |  |  |
| --- | --- | --- | --- |
| 英雄类中定义的属性包括炸弹数量，子弹类型等，全部属性如表23・21所示。  <23-21英雄类的属性 | | | |
| 序号 | 名 称 | 说 | 明 |
| 1 | is\_power | 是否无敌，刚登场有3 s无敌时间 | |
| 2 | bomb\_count | 炸弹数量，默认携带3颗炸弹 | |
| 3 | bullets\_kind | 子弹类型：0——单排；1——双排 | |
| 4 | bullets\_group | 子弹精灵组，后续的碰撞检测需要使用 | |

英雄对象中同样需要重写父类方法，并根据游戏规则还需要增加2个方法，方法列表如 表 23-22 所TH。

表23-22英雄飞机类的方法

|  |  |  |
| --- | --- | --- |
| 序号 | 名 称 | 说 明 |
| 1 | reset\_plane | 重置飞机，英雄被撞毁后，重新设置飞机的属性并发送自定义事件 |
| 2 | update | 更新飞机，调用父类方法设置飞机的显示图像，然后根据叩date方法的参数修改英雄 的位置，并将英雄限定在游戏窗口之内 |
| 3 | blowup | 引爆炸弹，炸毁游戏窗口内部的所有敌机，并返回得分 |
| *4* | fire | 射击，创建3个子弹精灵并添加到子弹精灵组以及display group |
| 2实瑰薨繼类的箋竊訪能  接下来，就实现英雄飞机类的基本功能。首先在gamejtems.py模块的顶部定义全局常量 记录英雄默认炸弹数量，代码如下：  HERO\_BOMB\_COUNT = 3 # 英雄默认炸弹数量  *#英*臨默认石始位置  HERO\_DEFAULT\_MID\_\_BOTTOM = (SCREEN\_RECT.centerx, SCREEN\_RECT.bottom - 90) | | |

然后在game items.py模块中创建Hero类,代码如下所示。

class Hero(Plane):

*° u "*英雄类"° "

def init (self, ^groups):

"° °初始化方法

:param groups :要添加到的精灵组

super(). init

(1000, 5r 0f "me\_down.wavn ["me%d.pngn % i for i in "mel.png"r

[nme\_destroy\_%dopngn % i 大groups) self.is\_power = False self obomb\_count = HERO\_BOMB\_COUNT self.bullets\_kind = 0 self.bullets\_group = pygame.sprite.Group() #初始位置 ~

range(1, 3)]z

for

i in range(1, 5)]*《*

无敌标记 炸弹数量 子弹类型 子弹精灵组

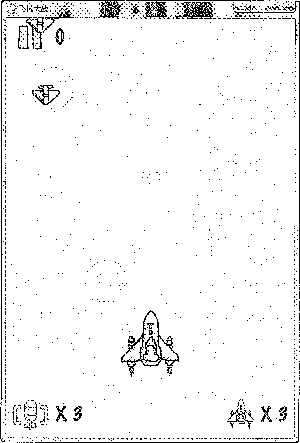


图23-40英雄飞机出现在窗口中间 靠下的位置

self.rect.midbottom = HERO\_DEFAULT\_MID\_BOTTOM

最后修改Game类的初始化方法，将原有使用Plane类创建英雄的代码，替换成使用Hero 类创建,并且使用英雄的炸弹数量属性设置指示器面板的显示，代码如下:

# 4.创建精灵

#背景精灵,交替滚动

self.all\_group.add(Background. (False), Background(True)) #指示器面板

self o hud\_panel = HUDPanel(self.all\_group)

#创建敌机

self„create\_enemies()

#英雄精灵 —

self »hero = Hero(self o all\_group)

#设置面板中炸弹数量 —

self o hud\_panel.show\_bomb(self.hero.bomb\_ count)

运行游戏，可以看到英雄飞机出现在游戏窗口中间靠 下的位置，并且在指示器面板中显示的炸弹数量是3,如 图23-40所示。

氟懊速移动英羅飞H

接下来实现使用键盘的方向键(3』、匸■、丄―』、 [->])在游戏窗口中快速移动英雄飞机的功能。之前已 经介绍过pygame监听键盘事件的方式*，*不过在这一小节， 要先给大家介绍另外一种处理方式。

(1)持续按键的处理

pygame专门针对持续按键的游戏开发需求，提供了一 种处理键盘的方式,即使用key模块提供的get\_pressed() 方法获得当前时刻的按键元组，然后使用按键常量作为元

570曷"Pjthoo薰臟编蔭 网零攣Python 组索引，判断某一个键是否被按下，按下则对应的值是L没按则对应的值是0。

接下来，在Game类的游戏循环中添加新的代码。要注意的是，游戏暂停或结束时，是不 允许移动英雄飞机的。调整后的代码如下列粗体字所示。

else:

self.hud\_panel.panel\_resums(self.all\_group)

#荻得当希诗刻的按键元组' —

keys = pygame.key.get\_pressed()

#判断是否按下有方向镜

if keys[pygame»K\_RIGHT]:

self.. hero <, rect. x 1= 10

#修改逐帧动画计数器

frame counter = (frame\_counter + 1) % FRAME\_INTERVAL

#更新all\_group中所有精灵内容

self . all\_\_group . update (f rame\_counter == 0)

运行游戏，如果按住右方向键不放，会发现英雄飞机会快速地从游戏窗口的右侧移出。

注意：

我们之前学习的监听键盘事件的方法，并不适合处理类似于让英雄飞机在游戏窗口内快速 移动这种需求。原因是什么呢？让我们用代码来验证一下。

首先注释掉上文中关于持续按键的代码，修改Game类的event handler方法,当监听到玩 家按下右方向键时修改英雄飞机的矩形区域，代码如下列粗体字所示。

#判断是否正在游戏

if not self. is\_game\_over and not self a is\_pau.se :

#监听玩家羸下字母"饥引爆炸弹 —

if event.type == pygame «KEYDOWN and event key == pygame »K\_b:

# TODO测试炸毁所有敌机

for enemy in selfoenemies\_groupasprites():

enemy o hp = 0

# TODO测试监听方向按键

if event o *type* == pygame KEYDOWN and event o key == pygame.K\_\_RIGHT: self o hero.rect o x += 10 1 - .

运行游戏，会发现尽管现在按下右方向键，已经能让英雄飞机向右移动了。但是，如果我 们想让英雄飞机连续地向右移动，就必须要抬起再按下右方向键才能够做到。这种交互方式， 在飞机大战这样激烈的射击类游戏中，显然不能满足玩家的需求。(请注意，要在测试完成之后， 将代码恢复到测试之前的代码，即依然使用pygame提供的持续按键处理方式。)

1. 持续按键的方向判断

之前已经介绍过，在getjressed方法返回的按键元组中，如果某一个键被按下，对应的值 要么是1,要么是0。而在飞机大战游戏中，英雄飞机的移动方向分为水平和垂直两种。那么， 怎样编写代码才能简化我们对方向的判断呢？

这里有一个小技巧,首先使用keys[pygame.K\_RIGHT] - keys[pygameX\_LEFT]计算水平移 动的基数，如以下代码所示。

#水平移动基数

move\_hor = keys[pygame.K\_RIGHT] - keys[pygame.K\_LEFT]

使用上面代码计算水平方向移动的基数，其原理如表23・23所示。

« 23-23计算英雄飞机水平方向移动基数的原理

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 用户操作 | keys[pygame.K\_R!GHT] | keys[pygame.K\_LEFT] | 移动基数 | 结果 |
| 不按键 | 0 | 0 | 0 | 不移动 |
| 按下右键 | 1 | 0 | 1 | 向右移动1 |
| 按下左键 | 0 | -1 | -1 | 向左移动1 |
| 按下左右键 | 1 | 1 | 0 | 不移动 |

按照pygame坐标系的设定，x轴沿水平方向向右，其数值逐渐增加。因此可以把移动基 数值作为英雄飞机水平移动的基数，1表示向右移动，T表示向左移动。然后，用移动基数值 乘以英雄飞机的速度，就可以计算得到水平方向需要移动的距离了。

同样的道理，可以使用类似的方法来计算英雄飞机垂直移动的基数，代码如下辭

#垂直移动基数

move\_ver = keys[pygame.K\_DOWN] - keys[pygame.K\_UP]

1. 移动英雄飞机

接下来，在Hero类中重写父类的update(\*args)方法，首先调用父类方法处理显示图像的 更新，然后增加代码修改飞机的矩形区域。

在这里需要注意的是，Sprite提供的update(\*args)方法是可以接收多值参数的，而之前我 们已经把第1个参数作为更新图像的标记。在Hfew类中，可以考虑把第2个参数作为水平移 动基数，第3个参数作为垂直移动基数。

重写后的update(\*args)方法代码如下:

def update(selfz \*args):

5”更新英雄的图像及矩形区域

:param args: 0 更新图像标记1水平移动基数2垂直移动基数

n n n

#调用父类方法更新飞机图像-注意args需要拆包

super () »update(\*args)

#如果没有传递方向基数或者英雄被撞毁，直接返回

if len(args) != 3 or self.hp <= 0: -

return

#调整水平移动距离

self.rect.x += args[1] \* self«speed

self.rect.y += args[2]火 self.speed

#限定在游戏窗口内部移动

self.rect o x = 0 if self.rect.x < 0 else self.rect.x

if self.rect.right > SCREEN\_RECT.right:

self.recteright = SCREEN\_RECT.right

self.rect.y = 0 if self.rect.y < 0 else self.rect» y

if self.rect.bottom > SCREEN\_RECT.bottom:

self.rect.bottom = SCREEN\_RECT.bottom

接下来修改Gmw类的游戏循环，在计算完移动基数和逐帧动画计数器之后，让all\_group 调用upda於方法并传递3个参数，代码如下所示，其中的粗体字就是修改过的代码。

self.hud\_panel.panel\_resume(self.all\_group)

#获得当新时刻的按键元鎏「

keys = pygameokey.get\_pressed()

#水平移动墓数

move\_hor = keys[pygame»K\_RIGHT] - keys[pygame»K\_LEFT]

#垂宣移动迭数 a

move\_ver = keys [pyganae o K\_DOW] - keys [pygame«K UP]

#修迅逐帧动画计数器 - 〜

frame\_counter = (frame\_counter + 1) % FRAME\_INTERVAL

#更新all\_group中所有精灵内容

self»all\_group o update (f rame\_countex == 0 f move\_hor f move\_ver)

运行游戏，现在已经能够在游戏窗口中灵活地移动英雄飞机的位置，并且移动范围已经被 限定在游戏窗口之中。

&炸靈游戏窗口内部的敌机 . .

接下来实现引爆炸弹的功能。为此a首先在Hem类中定义blowup方法，将出现在游戏窗 口内的敌机全部引爆，同时计算并返回得分。代码如下：

def blowup(selfz enemies\_group):

"n n引爆炸弹

:param enemies\_group: 敌机精灵组

:return :累计得分

# 1.如果没有足够数量的炸弹或者英雄被撞毁，直接返回 if self »bomb\_count <= 0 or self.hp <= 0: return 0

| self ..bomb\_count -= 1 | #炸弹数量-1 |
| --- | --- |
| score = 0 | #本次得分 |
| count = 0 | #炸毁数量 |
| # 2.遍历敌机精灵组，将游戏窗口内的敌机引爆 |  |
| for enemy in enemies\_groupssprites(): |  |
| #判断敌机是否进入藩戏窗口 |  |
| if enemy.rect«bottom > 0: |  |
| score += enemy.value | #计算得分 |
| count += 1 | #累计数量 |
| enemy o hp = 0 | #摧毁敌机 |

print (° 炸毁了 %d 架敌机，得分 %d° % (county score))

return score

要注意的是，敌机是垂直向下运动的，因此，当敌机矩形区域的bottom >0时，就说明敌 机已经飞入了游戏窗口。在程序中，我们只需要把这一部分敌机摧毁并且统计得分即可。

然后修改Game类的event\_handler方法，在监听到玩家按[B]键的事件处理中，去掉测 试代码，让英雄飞机调用blowup方法，并根据返回的分数，做后续的处理，包括更新游戏得分、 更新炸弹数量显示,以及判断是否升级到下一个关卡。如果升级,则调用create\_enemies方法 创建更多的敌机。代码如下：

#判断是否正在游戏

if not self o is\_game\_over and not self.is\_pause:

#监听玩家按下Kb『键，引爆炸弹 —

if event.type == pygame.KEYDOWN and event.key == pygame.K\_b:

#引爆炸弹

score = self «hero.blowup(self.enemies group) #更新炸弹数量显示 self.hud\_panel“ show\_bomb(self.hero.bomb\_count) #更新游我存分，如果关丰等级提升，创建新的敌机一 if self.hud\_panel» increase\_score(score):

self.create\_enemies()

运行游戏，按下键，可以看到游戏窗口中的所有 敌机都会被炸毁。如图23^41所示。

并且，观察控制台的输出就会发现，每次引爆炸弹， 并不会把敌机精灵组中的所有敌机全部炸毁，而只会把出 现在游戏窗口中的敌机炸毁。例如，某一次执行结果如下 所示。

炸毁了 8架敌机，得分8000

炸毁了 13架敌机，得分13000

炸毁了 15架敌机，得分15000

2X7

碰撞检测就是在每一次游戏循环执行时，检测游戏精 灵之间是否发生碰撞，例如：敌机碰到英雄飞机、子弹碰 到敌机等。碰撞检测在游戏开发中至关重要，其实现得好 坏直接影响玩家对游戏的体验及认可。

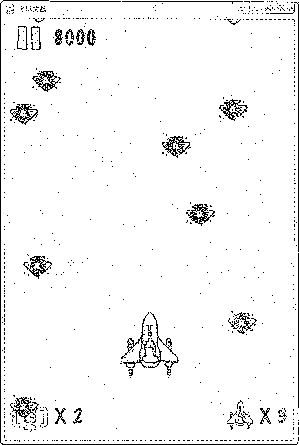
pygame的枣庆模块中提供了碰撞检测相关的方法，不过要想实现高品质的碰撞检测，还 需要配合使用pygame提供的另外一个模块，即mask模块。

图23-41按下[B] 屏幕上的

敌机都被炸毁

使用spri於模块提供的碰撞检测方法，能够非常容易地实现精灵间的碰撞检测。但是直接 使用这些方法并不能做到精细的检测，很难满足我们实际开发中的需求。在这一小节，就先来 介绍如何实现精细的碰撞检测。

1 B襪个KB検測方密

在spii於模块中提供了 2个非常重要的碰撞检测方法，分别是：

• spritecollide：检测某个精灵是否和某个精灵组中的精灵发生碰撞。

e groupcollide：检测两个精灵组之间是否有精灵发生碰撞。

这2个方法除了参数略有差异外，具体的使用以及实现的方式基本上都是一致的。具体介绍如下。 (1 ) spritecollide方法的语法如下所示。

spritecollide(sprite^ group, dokill7 collided = None) -> Sprite list

该方法有4个参数，分别介绍如下:

* sprite:表示要检测的精灵,如英雄飞机。

® group：表示要检测的精灵组，如敌机精灵组。

* dokill：表示是否移除金如果为True,会在检测到碰撞后，将gn)叩中与sprite发生碰撞 的精灵从group中移除。

® collided：表示用来检测碰撞的函数，如果传入None,则使用精灵的Ref属性来判断是 否发生了碰撞。

该方法的返回值是group中和sprite发生碰撞的所有精灵的列表。

(2 ) groupcollide方法的语法如下所示。

groupcollide(groupl, group2 z dokill1, dokill2 z collided = None) ->

Sprite\_dict

该方法有5个参数，分别介绍如下：

* group 1:表75要检测的精灵组1,如敌机精灵组。
* group2：表示要检测的精灵组2,如子弹精灵组。
* dokiill:表示是否从精灵组1移除9如果为Trus会将发生碰撞的精灵从group 1移除。
* dokili2:表示是否从精灵组2移除，如果为Tnw，会将发生碰撞的精灵从group2移除。
* collided：表示用来检测碰撞的函数，如果传入None,使用精灵的rect属性来判断是否 发生碰撞。

该方法的返回值是一个字典，字典的key表示group 1中检测到被碰撞的精灵，字典的 value是group2中与key发生碰撞的所有精灵的列表。

1. 碰撞検测的基本实现

了解了 pygame中的碰撞检测机制后，就可以使用spritecollide方法检测英雄飞机和敌机精 灵组之间的碰撞了。通过这次使用，可以学习碰撞检测方法的基本使用，并且了解直接使用这 个方法会存在哪些问题。

首先，在Game类初始化方法的末尾，增加以下测试代码，让所有创建出来的敌机都静止 在屏幕上，以方便我们观察碰撞检测的执行效果，代码如下：

# TODO:将所有敌机的速度设置为0,并修改敌机的初始位置 for enemy in selfoenemies\_group«sprites():

enemy.speed'= 0

enemy.rect.y += 400

self »hero o speed = 1

在测试代码中，让每架敌机矩形区域的y值增加400,使得敌机的初始位置和英雄飞机之 间有一段距离，将英雄飞机的速度设置为1,使得每次按下方向键移动的距离很小。这样，可 以方便我们使用方向键慢慢地靠近敌机，以观察碰撞检测的执行效果。

运行游戏，可以看到一大群小敌机静止在游戏窗口中等待着英雄飞机的碰撞。

接下来，在Game类中定义一个专门负责碰撞检测的方法check\_collide9并实现如下代码。

def check\_collide(self):

° ° "碰撞检测"° "

#「检测英雄飞机和敌机的碰撞 ：

collide\_enemies = pygame.sprite.spritecollide(self o herof

self.enemies\_group7

Falsef

None)

for enemy in collide\_enemies:

enemy hp = 0

#摧毁发生碰撞的敌机

然后在游戏循环中，恢复完指示器面板之后，调用刚刚实现的碰撞检测方法，如下列粗体 代码所示。

selfhud\_\_panel.panel\_resume (self。all group)

#碰饉检師 — —

self o che ck\_co11ide()

#获得当前日疚＞|的按键元组

keys = pygame » key s get\_pressed()

运行游戏，使用方向键慢慢地让英雄飞机靠近敌机，会发现每当英雄飞机即将要靠近敌机 时，敌机就会被撞毁，而不是两架飞机真正碰撞在一起，才产生碰撞效果，这是什么原因呢？

我们回顾一下spritecollide方法的最后一个参数collided,它表示用来检测碰撞的函数，如 果传入None,则使用精灵的rect属性来判断是否发生碰撞。

如图23-42所示，如果使用精灵的&ct属性做碰撞检测歹则意味着，只要两个精灵的矩形 区域发生重ft, pygame就会认为精灵之间发生了碰撞。也正因为如此,才会出现刚刚看到的情况， 每当英雄飞机即将要靠近敌机时，敌机就会被撞毁。这样的碰撞测试是不精确的，用户体验也 不尽如人意。

3町高品质的碰擄澄测

只使用碰撞检测得到的碰撞结果并不精确，我们有必要利用sprite模块提供的collide\_mask 方法来实现高品质的碰撞检测。按照pygame官方文档的说明，collide\_mask方法可以被作为参 数传递给碰撞检测方法，并且，如果在程序中需要做频繁的碰撞检测，应该在创建精灵时就为 精灵建立一个mask （遮罩），否则会影响程序的执行性能。

所谓mask （遮罩），可以把它理解为图像的轮廓填充，也就是先描边再填色。有颜色 的部分会被认为是精灵的实体部分，在做碰撞检测时没有颜色的部分会被忽略，如图23-43 所示。

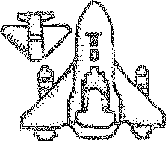
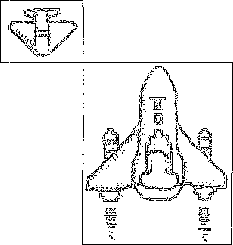


图23-42使用rect属性进行碰撞检测

图23-43使用了 mask后的碰撞检测

要实现高品质的碰撞检测，只需要将pygame.sprite.collide mask作为参数传递给碰撞检测 方法，然后在初始化精灵时，为精灵建立mask属性，以提高性能。

我们先修改上一小节完成的check\_\_collide法，验证能否实现精细的碰撞检测，修改后的代 码如下。

def check\_collide(self):

° ° "碰撞检测n ° "

# 1/检测英雄飞机和敌机的碰撞

collide\_\_enemies = pygame » sprite.spritecollide(self.hero

self.enemies\_groupf

False7

pygame。sprite。collide\_mask)

for enemy in collide\_enemies:

enemy. hp = 0 #摧毁发生碰撞的敌机

再次运行代码，移动英雄飞机进行测试，可以看到碰撞检测的精细度有了非常明显的提高。

然后在GameSprite类初始化方法的末尾,为精灵创建mask属性,代码如下:

#图像遮罩，可以提高碰撞检测的执行性能

self.mask = pygame.mask.from\_surface(self.image)

由于我们使用的所有飞机精灵和道具精灵都是由GameSprite或者它的子类创建的。因此， 只需要在这个父类的初始化方法中创建mask属性，所有子类对象就能够拥有了。

23.7.2 .

如果敌机在飞行途中与英雄飞机相撞，那么英雄飞机会被撞毁。如果还有剩余命数，那么 在英雄牺牲的位置出现新的英雄继续战斗。如果没有剩:余命数方游戏结束。而撞毁英雄飞机的 敌机，同样要播放被撞毁动画。动画播放完成后，敌机会被设置回初始状态，再次从游戏窗口 上方飞出加入战斗。接下来就来一一实现这些功能。

首先，删除之前在Game类初始化方法末尾增加的测试代码，让敌机恢复原有的飞行状态。 然后，修改Grille类的check\_collide方法，在该方法中判断英雄是否处于无敌状态，如果是， 不做碰撞检测；否则，检测英雄飞机和所有敌机的碰撞。如果撞到敌机，让英雄牺牲，撞到英 雄的敌机同样也会被撞毁。

修改后的check collide方法的代码如下所示。

#「检测英雄飞机和敌机的碰撞-如果英雄处于无敌状态，彼此不能碰撞

if not self.hero.is\_power:

enemies = pygame sprite . spritecollide (self .hero7

|  |  |
| --- | --- |
|  | self.enemies\_groupz  Falsef  pygame.sprite。collide\_mask) |
| #是否撞到敌机 |  |
| if enemies: |  |
| self.hero a hp = 0 | #英雄被撞毁 |
| for enemy in enemies: |  |
| enemy s hp = 0 | #敌机同样被撞毁 |

要注意的是，无敌状态的设定就是为了保证新英雄再次登场时不会立即被周边的敌机撞毁。 所以，在上述代码中，如果英雄正处于无敌状态，那么即使双方发生了碰撞，也不做任何的处理。 这样可以方便玩家在游戏中，使用方向键让英雄迅速移动到安全区域进行战斗。

再次运行游戏，英雄飞机已经能够和与之相撞的敌机一起爆炸了。

不过现在完成的游戏，存在一个问题，在敌机靠近英雄英雄飞机时引爆炸弹，然后迅速靠 近爆炸中的敌机，英雄英雄飞机会被敌机残骸撞毁。如果不希望英雄英雄飞机被敌机的残骸撞 毁，可以对碰撞检测之后的列表做一个过滤。在check\_collide方法中，添加如下粗体字所示的 代码。

enemies = pygame.sprite.spritecollide(self.hero7

self»enemies\_group

False, pygame.sprite.collide\_mask)

#过滤掉已经被摧毁的故机 ~

enemies = list (filter (lambda, x: > 0 enemies)) •

再次运行游戏，现在那些爆炸中的残骸已经无法撼动英雄飞机了。

2.谶帝襲勰飞机牺整通顒

按照游戏规则描述，应该在英雄飞机牺牲的动画播放完成之后，才能让新的英雄飞机登场 并更新游戏画面。

在我们之前完成的代码中，每当飞机的被摧毁动画播放完成之后，都会调用自己的龙 pla眼方法来重设飞机的数据。那么问题来了，我们并不能在Hero类的代码中直接更新游戏的 画面。

要解决这个问题，就需要使用到pygame在event模块中提供的post方法。由post方法发 布一个英雄牺牲的用户自定义事件，这样我们就可以在Game的事件监听方法中监听事件并且 更新游戏的画面了。

接下来,先在game\_items.py模块的顶部定义全局常量记录英雄牺牲事件代号,代码如下: HERO\_ DEAD\_EVENT = pygame . USEREVENT # 英雄牺牲事件

然后,在Hero类中重写父类的reset\_plane方法，并实现以下代码。

def reset\_plane(self): 叩""重置英雄° ° " #调用父类方法重置图像相关属性 super().reset\_plane() self.is\_power = False self »bomb\_count = HERO\_BOMB\_COUNT self.bullets\_kind = 0 #发布英雄牺牲孝件

#无敌标记

#炸弹数量

#子弹类型

pygame.event.post(pygame.event.Event(HERO\_DEAD\_EVENT))

要注意的是，在重置英雄的方法中，暂时将无敌标记sel£is\_\_power属性设置为False,这样 可以方便稍后的代码测试。

最后，修改Game类的event handler方法,增加英雄牺牲事件的监听,一旦监听到英雄牺 牲，需要修改生命计数以及更新生命计数和炸弹数量的显示，增加的代码如下列粗体字所示。

#判断是否正在游戏

if not self o is\_game\_over and not self.is\_pause:

#监听英雄牺霆事件~ ~

if ©vent»type == HERO\_DEAD\_EVENT : print (”英雄牺牲了 .…“) #生命计数—1 self«hud^panel. lives\_coun't -= 1

#更新生命计数显示 "

self.hud\_panel. sliow\_lives ()

#更斷炸弾显示 a

self.h.ud\_panel. show\_\_\_boafc (self .hero «bomb\_\_count)

#监听玩家按KbS键，引爆炸弹

if event.type == pygame oKEYDOWN and event.key == pygame.K\_b:

运行游戏，一旦英雄牺牲后，游戏窗口中的数据显示已经正确了，同时也能正确地判断游 戏结束。

3。设置英雄无敌时间

目前实现的代码中，由于还没有设定英雄的无敌状态，所以一旦英雄再次登场，在周围敌 机非常多的情况下，英雄几乎来不及做出任何的反应就会再次被撞毁「我们有必要为英雄飞机 设置无敌时间。

接下来，就利用定时器事件来处理英雄无敌的设定。首先，在game\_items.py模块的顶部 '定义全局常量记录取消英雄无敌的定时器事件，代码如下。

HERO\_POWER\_OFF\_EVENT = pygame . USEREVENT + 1 # 取消英雄无敌事件

然后,在Hero类的reset\_plane方法末尾设定定时器事件，代码如下。

#设置3 s之后取消无敌定时器事件

pygame。time.set\_timer(HERO\_POWER\_OFF\_EVENTz 3000)

修改之前代码中英雄的无敌标记，将selfisjower的值设置为True,修改后的代码如下所示。 self . is\_power = False # 无敌标记

最后，修改Game类的event\_handler方法，增加取消英雄无敌事件的监听，一旦监听到取 消英雄无敌，需要修改英雄的无敌属性并且关闭定时器，增加的代码如下列粗体字所示。

#判断是否正在游戏

if not self.is\_game\_over and not self.is\_pause:

#监听取消縁无— if event <> type == HERO\_POWER^OFF\_EVENT :

print (n取消无故状态…")

#设置英雄屬性

self.hero is\_power = False

#取消定时器 pygame. time. set\_\_timer (HERO^POWEIR\_OFF\_EVENT, 0)

#监听英雄牺牲事件 ~ ~ ~

if event»type == HERO\_DEAD\_EVENT:

需要注意的是，在使用SRtJmw方法时，如果给时间间隔参数设置为0,会关闭该定时器事件。 运行游戏，会发现每次英雄飞机被撞毁之后，无敌状态只会持续3 s。但是，当英雄飞机 没有剩余命数，整个游戏结束之后，玩家按空格键开启新游戏时，英雄飞机会从上一次牺牲的

第23章 飞机大fit Q 579 位置开始新游戏，这一点显然不够合理。

为此，我们修改Gmw类的reset\_.game方法，在方法末尾重新设置英雄属性，并且指定英 雄的初始位置，代码如下：

#设置英雄的初始位置

self o hero.rect.midbottom = HERO\_\_DEFAULT\_MID\_BOTTOM

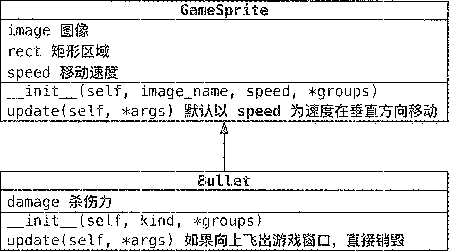
再次运行游戏，现在每次游戏结束后，玩家按下空格键开启新游戏，英雄飞机已经能够回

到初始位置开始新的战斗了。

23.7.3靈羅賤鮒琴彎 •

在这一小节中，我们完成英雄发射子弹的功能，主要包括设计子弹类、实现英雄发射子弹 的功能，以及实现子弹击中敌机的功能。

根据游戏规则，我们设计子弹类(即侦)，其类图如图23W所示。



K 23=44子弹类的设计

从类图可知，子弹类中添加了一个属性damage,表示子弹的杀伤力，默认为"

另外,在子弹中重写父类的update方法。在该方法中先调用父类方法,让子弹以speed为

速度在垂直方向上运动，然后进行判断，一旦子弹从游戏窗口上方飞出身就销毁子弹精灵对象。

接下来，就按照刚刚明确的思路，在game\_items.py模块中创建继承自GameSprite类的 Bullet类,实现代码如下所示。

class Bullet(GameSprite):

n h n n n n

def init (self, kind, ^groups):

”5初始化方法

:param kind: 子弹类型

:param groups :要添加至”的精灵组

H H H

image\_name = "bulletl.png" if kind == 0 else nbullet2.png" super () o init (image\_\_namez -12 7 \*groups) self o damage = 1 # 杀伤力

def update(self, \*args):

super () . update (\*args) # 向上移动

#判断是否从上方飞出窗口

if self.rect.bottom < 0:

self.kill()

在实际编写代码时，将子弹的速度设置为T2可以表示向上方飞行。当子弹从上方飞岀游 戏窗口后，子弹的使命就终结了，此时可以使用Sprite类封装的kill方法，将子弹精灵从所有 精灵组中移除，及时释放内存。

2.实现英雄发射子弹的功能

接下来，就在Hero类中定义嵌方法，并利用定时器事件，实现英雄发射子弹的功能。 首先，在gamejtems.py模块的顶部定义全局常量记录英雄发射子弹的定时器事件，代码如下: HERO\_FIRE\_\_EVENT = pygame . USEREVENT + 2 # 英雄发射子弹事件

然后，在Hero类的初始化方法的末尾，设置英雄发射子弹的定时器事件，代码如下： self .bullets\_kind = 0 # 子弹类型

self. bullets\_group = pygame , sprite . Group () # 子弹精灵组

#初始位置

self.rect.midbottom = HERO\_DEFAULT\_MID \_BOTTOM

#设置0.2 s发射子弹定时器事作 ~ ~

pygame.time.set\_timer(HERO\_FIRE\_EVENTz 200)

下一步,在Hero类中定义fire方法,并实现以下功能:

(1 )根据bullets\_kind属性，连续创建子弹精灵。如果bullets\_kind = 0 ?创建3颗子弹； 如果bullets\_kind = 1,创建6颗子弹，每两颗子弹一排。

(2 )根据bullets\_kind属性，设置子弹精灵的初始位置在英雄飞机的正上方。

要注意的是，子弹精灵需要被添加到两个精灵组:bullets\_group (用于后续的碰撞检测) 和display\_group (用于精灵的显示)。

fire方法的代码如下:

def fire (self 7 display\_group):

5W发射子弹

:param display\_group :要添加的显示精灵组

#需要将子弹精灵添加到两个精灵组

groups = (self.bullets\_groupf display\_group)

#测试子弹增强效果 - -

# self obullets\_kind = 1

for i in range(3):

#创建子弹精灵

bulletl = Bullet (self.bullets\_kinclA \*groups)

#计算子弹的垂直位置 —

y = self. rect <, y - i \* 15

#判断子弹类型

if self.bullets kind == 0:

bulletl. rect. midbottom = (self:, rect. centerx, y) else :

bulletl.rect.midbottom = (self.rect.centerx - 20, y)

#再创建一颗子弹

bullet2 = Bullet(self.bullets\_ kindz ^groups)

bullet2.rect.midbottom = (self.rect.centerx + 20 z y)

最后，修改Game类的event\_handler方法，增加英雄发射子弹事件的监听。一旦监听到事

件，让英雄飞机调用如e方法发射子弹，新增代码如下列粗体字所示。

#判断是否正在游戏

if not self.is\_game\_over and not self » is\_pause:

#监听发射科夢" -

if event t== HERO\_FIREmSVENT :

selfohero»fire(selfoall\_group)

#监听取消英雄无敌事件 …

if event.type == HERO\_POWER\_OFF\_EVENT:

运行游戏可以看到，每隔02 s会从英雄飞机的头部向上方连续发射子弹了。

3.实现子弹击中敌机的功能

实现了英雄发射子弹的功能之后，接下来9就对Game类的check\_collide方法进行扩展° 实现子弹击中并摧毁敌机的功能。

之前我们曾经使用过sprite模块提供的spritecollide方法检测英雄飞机和敌机之间的碰撞， spritecollide方法可以方便地实现一对多关系的碰撞检测，即一架英雄飞机对多架敌机。

而在这一小节，我们需要处理的是多对多关系的碰撞检测，即多发子弹对多架敌机。针对 这种需求，sp庆模块提供了另外一个方法groupcollideo它能够方便地检测敌机精灵组和子弹 精灵组中的精灵是否发生了碰撞。如果检测到碰撞，会返回一个字典，其中key是gwupl中检 测到被碰撞的精灵，value是group2中与key发生碰撞的所有精灵的列表。

由于在实现子弹击中敌机这个功能时，既要遍历碰撞字典，又要遍历字典内部的子弹列表, 所以在开始编写代码之前，先使用一个流程图明确思路，如图23N5所示。

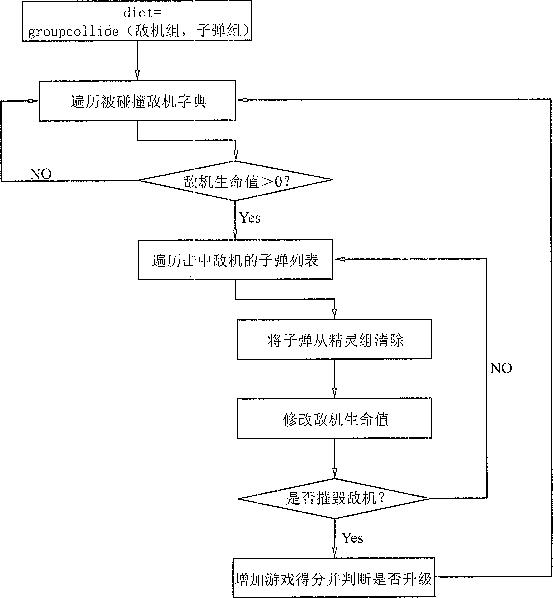


图23-45子弹击中敌机的流程

我们按照流程图，在Game类的check\_collide方法的末尾增加子弹击中敌机的处理，代码 如下：

# 2。检测敌机被子弹击中

hit\_enemies = pygame.sprite groupcollide(self.enemies\_groupr self.hero »bullets\_groupf Falsex

Falsef pygame.sprite*。*collide\_mask)

#遍历字典

for enemy in hit\_enemies:

#已经被摧毁的萩机不需要浪费子弹

if enemyohp <=0:

continue

#遍历击中敌机的子弹列表

for bullet in hit\_enemi.es [enemy]:

* 1>将子弹从所看精灵组中清除

bullet.kill()

* 2〉修改敌机的生命值

enemy.hp -= bullet.damage

* 3>如果敌机没有被摧毁，继续下\_颗子弹

if enemy.hp > 0:

continue

* 4〉修改游戏得分并判断是否升级

if self o hud\_panelo increase\_score(enemy value):

self.create\_enemies()

* 5>退出遍历子弹利表循环

break

运行游戏，现在已经可以进行非常激烈的对战了。但是，如果英雄飞机闯到第三关牺牲之后, 玩家按下空格键开启新游戏，新游戏中的敌机仍然是第三关的配置，而不是按照游戏规则设定 的第一关只有小敌机。

为此，修改Game类的reset\_game方法,在方法末尾清空所有的敌机以及之前英雄飞机残 留的子弹，然后再重新创建敌机，代码如下：

#清空所有敌机

for enemy in self . enemies\_\_group : enemy.kill()

#清空残留子弹

for bullet in self <. hero o bullsts\_group : bullet.kill()

#重新创建敌机

self.create\_enemies()

再次运行游戏，现在每次开启新一轮游戏时，都会按照游戏规则的设定，从第一关的敌机 配置开始顺序登场了。

23q7d4薰鑑籲麒道聂 •

游戏开始后，每隔30$会从游戏窗口上方随机位置向下飞出游戏道具，其中包括炸弹补给 和子弹增强道具。在这一小节中，就实现英雄拾取道具的功能。

第23章 飞机大战 £ 583 牝设计道具美

在设计'道具类之前，首先要明确程序中道具位置的处理方式，如图23・46所示。

按照游戏设计我们知道，游戏中有且仅有两个道具：炸弹补给和子弹增强。为了避免在程

序中反复地创建相同道具精灵，可以按照以下思路来处理：

(1 )在游戏初始化时创建2个道具精灵,并且把道具精灵的初始位置设置在游戏窗口的下方。

(2 )游戏开始后，每隔30 s调用道具精灵的投放炸弹的方法，将道具精灵设置到游戏窗 口上方的随机位置，准备开始垂直向下运动。

(3 )在道具精灵向下方运动时，如果与英雄飞机相遇，则设置相关属性，然后将道具精 灵的位置设置到游戏窗口的下方。

(4)道具精灵处于游戏窗口下方时，不再更新道具精灵的位置。

根据上述分析，进行道具类(Supply)的设计，其类图如图23N7所示。

投放道具位置

00*專鵬勰年*

i~~~-i…5f 「 仓盘 01鳳 .

image SB rect矩形区域

update (self, \*args)歓认以s腿为速度在蠡専方価覺袍

、 Supply

布3薄节礴福1子弹增强) 俯音效名

throw„supply(self)殺放道具 updg( 的「g提 更新位置

图23・47道具类的类图

道具初始和终止位置

图23-46道具的初始和终止位置

道具类中封装了 2个属性，分别是：

。kind：道具类型，0值代表炸弹补给，1值代表子弹增强。

• wav \_name:投放道具时播放的音效文件名。

另外，在道具中定义一个方法throw\_supply?并重写了父类的update方法，介绍如下：

。throW\_supply方法：投放道具，将道具设置到游戏窗口上方的随机位置。

' update方法:更新道具位置，如果处于游戏窗口下方，不再更新位置。

接下来，就按照刚刚明确的思路，在gamejtems.py模块中创建从GameSprite类派生的 Supply类,并实现以下代码。

class Supply(GameSprite):

H n h道^具类° ° n

def init (self7 kind, \*groups):

° ° °初始化方法

:param kind: 道具类型

:param groups :要添加到的精灵组

n n n

#调用父类方法

image\_name = "%s\_supply.png° % ("bomb" if kind == 0 else "bullet") super() o init (image\_namez 5, ^groups)

#道具类虹 ~~ ~

self.kind = kind

#音频文件名

self.wav\_name = "get\_%s.wavH % (°bornb" if kind == 0 else "bullet") #初始位亶 —

self .rect <,y = SCREEN\_RECT . h

def throw\_supply(self):

° ° w投放道具° ° °

self.rect.bottom = 0

self.rect.x = random.randint(07 SCREEN\_RECT.*w -* self.rect.w) def update(self, \*args):

"5更新位置，在屏幕下方不移动""°

if self.rect.h > SCREEN\_RECT.h:

return

#调用父类方法，沿垂直方向移动位置

super().update(\*args)

2軍定踢疆敏道具

接下来，就利用定时器事件实现一下定时投放道具的功能。

首先，在gamejtems.py模块的顶部定义全局常量记录投放道具的定时器事件，代码如下： THROW\_SUPPLY\_EVENT = pygame . USEREVENT + 3 # 投放道具事件

然后,在Game类中定义create\_supplies方法,创建2个道具精灵，并设置定时器事件， 代码如下。

def create\_supplies(self):

° ° "创建道具« "»»

Supply (0z self . supplies\_groupf self, a].l\_group)

Supply(1, self.supplies\_groupz self.all\_group)

#设置30s投放道具定时器事冲(测试时用10s)— pygame . time . set\_timer (THROW\_SUPPLY\_\_EVENTz 10000)

要注意的是，在上述代码中，为了方便测试，将代码中投放道具的间隔时长设置成了 10 so 在测试完成后，要将间隔时长设置为正确值30 so

接下来9在Game类的初始化方法的末尾,调用create\_supplies方法，代码如下:

#创建道具

self.create\_supplies()

最后，修改类的event\_hand辰方法，增加投放道具事件的监听，一旦监听到事件， 从道具精灵组随机出现一个道具9然后让道具调用tlirow\_supply方法开始投放，代码如下：

#判断是否正在游戏

if not self.is\_game\_over and not self.is\_pause:

#监听投放道具事件

if event.type == THROW\_SUPPLY\_EVENT: supply = random.choice(self.supplies\_grouposprites()) supply “ throw\_supply()

#监听发射子弹事件-

if event .type == HERO\_\_FIRE\_ EVENT :

运行游:戏，可以看到，每隔10s会从游戏窗口的上方出现一个随机道具快速地掠过游戏窗口。 3.爨絶拾取道具 .

接下来，就对Game类的check\_\_collide方法进行扩展，实现英雄拾取道具的功能。

在这一小节，我们仍然使用sprite模块提供的spritecollide方法检测英雄飞机和道具之间的 碰撞。

首先，为了方便碰撞检测的代码编写，我们在game\_items.py模块的顶部定义全局常量记 录关闭子弹增强的定时器事件，代码如下：

BULLET\_ENHANCED\_OFF\_EVENT = pygame . USEREVENT + 4 # 关闭子弹增强事件

接下来，在Gmne类check\_collide方法的末尾增加英雄拾取道具的处理，代码如下言

# 3. •荚雄拾取道具 supplies = pygame.sprite.spritecollide(self.hero,

self.supplies\_groupf

False,

pygame.sprite.collide\_mask) if supplies:

supply = supplies[0]

#将道具设置到游戏窗口下方

supply.rect.y = SCREEN\_RECT.h

#判断道具类型 一

if supply e kind == 0: # 炸弹补给

self.hero.bomb\_count += 1 self.hud\_panel.show\_bomb(self.hero.bomb\_count)

else: ~ #设置子葬增强

self.hero.bullets\_kind = 1

#设置关闭子弹增强的泛时器事件

pygame•time.set\_timer(BULLET\_ENHANCED\_OFF\_EVENT, 8000)

要注意的是，为了方便测试，将代码中关闭子弹增强的时长设置成了 在测试完成后，

应将时长更改为游戏设计中规定的20 So

最后，修改类的event\_handler方法，增加关闭子弹增强的监听，一旦监听到事件， 恢复子弹类型并且关闭定时器事件身如下列粗体字代码所示。

#判断是否正在游戏

if not self.is\_game\_over and not self.is\_pause:

#监听关闭孑潔瓚强争件 —

if event, type == BULLET\_EWHMTCED\_OFF\_EVEWT: self. hero <, bullets^kind = 0 pygame. time • set\_timer (BULLET\_\_ENHANCED\_OFF\_EVENT ? 0)

#监听投放道具事件 ~

if event.type == THROW\_SUPPLY\_EVENT:

运行游戏，可以看到英雄拾取到道具之后的效果。拾取炸弹补给后，界面上的炸弹数量会 加1,拾取子弹增强后，英雄飞机发射的子弹变成双排。

羽。區啬陳獅啬數

游戏又被称为第九种艺术，是各项艺术的集合体。在一个独立且完整的游戏世界中，音乐 是不可或缺的因素。游戏音乐可以通过声音来提升玩家对游戏的体验。通常大型游戏中游戏音 乐划分的类别比较细致，如主题曲、原声音乐、背景音乐及游戏音效等。要开发一款优秀的小 游戏，背景音乐和游戏音效这两种声音是必不可少的。

背景音乐可以烘托游戏的氛围，增强玩家的代入感。背景音乐是一个完整的音乐片段，一 般会被循环播放，玩家在整个游戏过程中始终都能听到。

游戏音效可以用于点缀或加强某一个游戏操作或事件，例如：发射子弹、投放道具以及飞 机被撞毁等。它的特点是播放声音较短，表现形式单一，但简洁有力，通常会在游戏中频繁播放。

为了在游戏中方便地播放音乐和音效，pyganw提供了两个模块，分别介绍如下：

° pygame.mixermusic模块：提供长音乐播放相关的功能。

° pygame.mixer：提供短音效播放相关的功能。

在这一小节*，*先来了解如何使用这两个模块提供的方法来播放音乐和音效。

们播放音乐

pygame.mixeEmusic模块提供一系列的方法，可以方便地控制长音乐的播放，常用方法如 表23-24所示。

表23-24 pygame.mixer.music模块的常用方法

|  |  |  |
| --- | --- | --- |
| 方 法 | 说 | 明 |
| load(音乐文件路径) | 从磁盘加载音乐文件，以准备播放 | |
| play(循环次数) | 开始播放，循环次数为-1,会一直循环播放: | ;循环次数为1,会播放1次后，再播放1次 |
| stop | 停止播放 | |
| pause | 暂停播放 | |
| unpause | 取消暂停，继续播放 | |
| set\_volume（音量） | 设置音量，音量参数的范围是0.0 ~1.0 | |

要在程序中播放背景音乐，通常只需要两个步骤：

(1 )使用load方法加载音乐文件。

(2 )使用play(4)方法循环播放。

接下来，就在Game类的初始化方法的末尾，增加以下测试代码，播放游戏的背景音乐, 代码如下。

# TODO测试-音乐和音效

* 1＞加载背景音乐文件准备播放

pygame omixer.music.load("。/res/sound/game\_music.ogg")

* 2＞播放音乐

pygame.mixer.music.play(-1)

运行游戏，玩家已经可以在背景音乐的伴奏下，开始激烈的空战了。

2«播放音效 •

pygamejnixRE模块专门提供了一个声音Sound类，可以创建声音对象并播放游戏音频。

|  |  |
| --- | --- |
| Sound类提供的常用方法如表23-25所示。  表23-25 Sound类的常用方法 | |
| 方 法 | 说 明 |
| Sound(音效文件路径) | 加载音效文件并创建声音对象 |
| sound.play() | 播放声音对象的音效 |
| sound.stop() | 停止声音对象的播放 |
| sound.set \_volume(音一量〕) | 设置声音对象的音量,音量参数的范围是0.0〜1.0 |

要在程序中播放音效，通常也只需要两个步骤：

(1 )创建并记录Soinid对象，注意：不同的音效需要创建不同的对象。

(2 )在需要时让Sound对象调用play()方法播放音效。

接下来，就在Game类的初始化方法的末尾，增加以下测试代码，播放英雄飞机被摧毁的 音效，代码如下：

* TODO测试-音乐和音效
* 3＞创建声音对象

hero\_down\_sound = pygame.mixer。Sound(°./res/sound/me\_down。wav") hero\_down\_sound.play()

运行游戏，在游戏刚刚启动时，能够听到口彭的一声爆炸音效。

要注意的是，由于游戏音效的声音大多是非常短促的，所以在游戏开发时，通常不需要考 虑停止正在播放的音效。

23.8.2谩讲舞靈飆置陳微鮑器鬟

＞此电鮑*；*虚面 ＞ 飞机大战＞ res 7- sound

通过前面章节的介绍我们知道，在pygam©中无论是播放音 乐还是音效都还是比较简单的。在游戏中，背景音乐只有一首 可以循环播放，游戏音效却有很多，而且每一个游戏音效都需 要单独创建一个对象，才能够在需要时播放。

•皀 bidfetwav

:從 button.wav

enecny! down.wsv

忘 enemy2 down.wsv

■0： enemyS^doyy'n.wav

en 创理 3 甌

:game\_music.ogg 卜 一~ 背景音乐

飞机大战游戏需要使用的音乐和音效文件都保存在res/ sound目录下,如图23-48所示。

suppiy.wav  
upgrade.wav  
*择* use bomb.wav

1. 皤放器类

为了简化在游戏中对音乐和音效播放的控制，我们设计一

个音乐播放器类，取名为MusicPiayer,其类图如图23・49所示。 图23-48项目中用到的背景 音乐和音效

. 叶koixPlaye 旷 *；*

soimct-dict音效字典

\_init\_(self, music\_file)加载背景音乐，创建音效对象字典 play\_sound(selfwav\_name)播放音效

play顼usix()繼放背景音乐

pau$$ic (i5\_pau$ed)暂停/恢复背景登乐

图23-49音乐播放器的类图

在MusicPlayer类中定义了一个属性,名*为*sound\_dict9表示音效字典,其中key是文件名, value是对应的Sound对象。

MusicPlayer类中定义的方法如表23-26所示。

> 23-26 MusicPlayer类中定义的方法

|  |  |  |
| --- | --- | --- |
| 名 称 | 说 | 明 |
|  | 参数music\_file是背景音乐文件名，res/sound目录下的其他文件都认为是音效文件 | |
| play\_sound | 播放音效名参数对应的游戏音效 | |
| play\_music | 静态方法，播放背景音乐 | |
| pause\_\_music | 静态方法，暂停/恢复背景音乐 | |

1. 加载和播放背景音乐

音乐播放器类的设计完成之后，就可以代码实现了。我们首先在game\_music.py模块中定 义并且实现MusicPlayer类，然后实现背景音乐的播放和暂停。

首先在文件顶部导入需要使用的模块，代码如下：

import os #需要遍历res/sound目录下的文件

import pygame

然后定义MusicPlayer类，并实现加载及播放背景音乐相关的方法，代码如下：

class MusicPlayer(object):

° n "音乐播放器类° ° ° res\_path = ° . /res/sound/" # 声音资源路径

def init (self7 music\_file):

° 5初始化方法

:param music\_\_file :背景音乐文件名

H „ H

# 1加载背景音乐

pygame 8mixer omusic . load (self . res\_path + music\_file) pygame .mixer »music «, set\_volume (0。2)

@staticmethod

def play\_music(): pygame.mixer.music.play(-1)

@staticmethod

def pause\_music(is\_pause):

if is\_pause:

pygame.mixer.music.pause()

else:

pygame.mixer.music.unpause()

要注意的是，代码中将背景音乐的音量设置成了 02?这是因为背景音乐文件的音量太大了， 我们把它降低一些，这样就不会干扰到稍后测试音效的播放了。

接下来，在Game类的初始化方法的末尾，删除掉之前的测试代码，增加以下代码，创建 音乐播放器并且循环播放背景音乐，代码如下：

* 5.创建音乐播放器

self.player = MusicPlayer(°game\_music.ogg")

self.player.play\_music()

最后，修改Game类的event\_handler方法，在监听到玩家按下空格键暂停或恢复游戏的同 时，暂停或恢复游戏背景音乐的播放，代码如下列粗体字所示。

elif event.type == pygame.KEYDOWN and event.key == pygame.K\_SPACE:

if self. is\_game\_over: # 游戏已经结束

self.reset\_game()

else : #切换暂停状态

self.is\_pause = not self.is\_pause

#暂停或稜复背景音乐 ~

self^playerepause\_music(self.is\_pause) ' ‘

运行游戏，游戏启动后仍然会循环播放背景音乐。如果在游戏执行过程中，按下空格键暂 停或者继续游戏9就会听到背景音乐的播放同样也会及时地被暂停或者恢复。

1. 加载和播放音效

背景音乐的功能实现以后，下一步就是按照音乐播放器类的设计，实现加载和播放音效的 功能。在这一小节需要播放的音效包括发射子弹、引爆炸弹、投放和拾取道具、敌机爆炸和英 雄爆炸。

首先，在MusicPlayer类的初始化方法的末尾增加代码，从./res/soimd @录加载所有音频 文件，并且将创建的声音对象，添加到sound\_dict音效字典中，代码如下？

* 2.加载音效字典
* 1＞定义音效字典属性

self.sound\_dict = {}

* 2＞获取目泉下的文件列表

files *= os* . listdir (self . res\_path)

* 3＞遍历文件列表

for file\_name in files:

#排除背景音乐

if file\_name == music\_file:

continue

#创建声音对象

sound = pygame.mixer.Sound(self.res\_path + file\_name)

#添加到音效字典，使用文件名作为字典的key ~

self. sound\_dict [file\_name] = sound

然后,在MusicPlayer中定义play\_sound方法,实现播放音效的功能，代码如下:

def plsiy\_sound (self 7 wav\_name):

° ° 播放音效

:param wav\_name : 音效文件名

self o sound\_dict[wav\_name]。play()

接下来回到Game类，按照游戏设定，逐一调用播放音效的方法。

1. 发射子弹

在event\_handler方法中找到监听发射子弹事件的分支，添加播放“发射子弹”音效的代码, 如下列代码中的粗体字所示。

#监听发射子弹事件

if event.type == HERO\_FIRE\_EVENT:

self。player play\_\_sound ("bullet .wavn)

self . hero .fire (self . all\_group)

1. 引爆炸弹

在event\_handler方法中找到监听玩家按[B ]键,引爆炸弹的分支,添加播放“引爆炸弹” 音效的代码，如下列代码中的粗体字所示。

#监听玩家按 定』键，引爆炸弹

[if event.type == pygame](#bookmark3089" \o "Current Document) [a](#bookmark3089" \o "Current Document) [KEYDOWN and event.key == pygame » K\_b: 課如果荚嫌没有牺犊同時有炸弾 —](#bookmark3089" \o "Current Document)

if self o hero o hp > 0 and self. hero. bomb\_\_count > 0 :

self o player. play\_sound ("use \_k>omb。wav")

#引爆炸弹 ; ~

score == self.hero o blowup(self » enemies\_group)

1. 投放和拾取道具

在event\_handler方法中找到监听投放道具事件的分支，添加播放"投放道具"音效的代码, 如下列代码中的粗体字所示。 '

#监听投放道具事件

if event.type == THROW\_SUPPLY\_EVENT:

self. player. pla]f\_sound (" supply. wav")

supply = random„choice(self.supplies \_group.sprites()) supply.throw\_supply()

然后，在check\_collide方法中，找到英雄拾取道具部分的代码，添加播放“拾取道具”音 效的代码，如下列代码中的粗体字所示。 1

if supplies:

supply = supplies[0]

#播放使殉道具音魏 —

self .player .pla^f\_sound (supply „wav\_name)

粗体字代码中，wav\_\_naine属性是之前设计道具类时定义的。

1. 敌机爆炸和升级

在check\_collide方法中,找到检测敌机被子弹击中部分的代码，调整代码如下:

#遍历击中敌机的子弹列表

for bullet in hit\_enemies[enemy]:

#省略部分代码二……

# 4>修改游戏得分并判断是否升级

if self ohud\_panel„increase\_score(enemy.value):

#播放升亙音效 -

self„player»play\_sound("upgrade.wav")

self 8 create\_enemies()

#播放敛机炸弹音效'

self • player。play\_\_sound (enemy. wav\_\_name)

# 5＞退出遍历子弹列表循环 ~

break

上述代码中的粗体字分别播放了升级音效和敌机炸弹音效。要注意的是，其中的wav\_ name属性是之前设计飞机类时定义的。

1. 英雄爆炸

在check\_collide方法中，找到检测英雄飞机和敌机的碰撞部分的代码，添加播放“英雄爆 炸”音效的代码，如下列代码中的粗体字所示。其中，wav\_nanie属性是设计飞机类是定义的。

#是否撞到敌机

if enemies:

#播放英雄被撞毁音效

self o player.play\_sound(self.hero.wav\_name)

self . hero . hp = 0 #英雄被撞毁

运行游戏，大家可以自行感受，在增加了游戏音效之后，飞机大战这款游戏的用户体验有 了一个明显的提升。

2X9项剧俸金

至此，我们已经完成了飞机大战项目的所有功能，通过这个游戏项目的设计和实现，体验 了使用Python语言开发一个桌面游戏项目的完整过程*，*并且达到了以下目的：

(1)体验面向对象设计和开发思想，将整个项目的功能封装为若干个特定的类，每个类 负责一个独立的功能。

(2 )体验pygame的高级功能。例如，使用key模块提供的gRt\_press©d()方法实现持续按 键功能，为精灵对象添加遮罩实现高精度的碰撞功能，为游戏添加声音和音效等功能，这些高 级功能都对提高游戏的可玩性，增强用户体验起到了巨大的作用。